


2012

Towards Evolving More Brain-Like Artificial Neural Networks

Sebastian Risi
University of Central Florida

 Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Risi, Sebastian, "Towards Evolving More Brain-Like Artificial Neural Networks" (2012). *Electronic Theses and Dissertations, 2004-2019*. 4675.
<https://stars.library.ucf.edu/etd/4675>

TOWARDS EVOLVING MORE BRAIN-LIKE ARTIFICIAL NEURAL NETWORKS

by

SEBASTIAN RISI

Diplom. Philipps-Universität Marburg, Germany 2007

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2012

Major Professor: Kenneth O. Stanley

© 2012 SEBASTIAN RISI

ABSTRACT

An ambitious long-term goal for neuroevolution, which studies how artificial evolutionary processes can be driven to produce brain-like structures, is to evolve neurocontrollers with a high density of neurons and connections that can adapt and learn from past experience. Yet while neuroevolution has produced successful results in a variety of domains, the scale of natural brains remains far beyond reach. In this dissertation two extensions to the recently introduced Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) approach are presented that are a step towards more brain-like artificial neural networks (ANNs). First, HyperNEAT is extended to evolve plastic ANNs that can learn from past experience. This new approach, called *adaptive HyperNEAT*, allows not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of arbitrary local learning rules. Second, *evolvable-substrate HyperNEAT* (ES-HyperNEAT) is introduced, which relieves the user from deciding where the hidden nodes should be placed in a geometry that is potentially infinitely dense. This approach not only can evolve the location of every neuron in the network, but also can represent regions of varying density, which means resolution can increase holistically over evolution. The combined approach, *adaptive ES-HyperNEAT*, unifies for the first time in neuroevolution the abilities to indirectly encode connectivity through geometry, generate patterns of heterogeneous plasticity, and simultaneously encode the density and placement of nodes in space. The dissertation

culminates in a major application domain that takes a step towards the general goal of adaptive neurocontrollers for legged locomotion.

*Ich widme diese Dissertation meiner Familie, insbesondere meinen Eltern, meiner Schwester
Jessica, meiner Oma Sigrid, meiner Großtante Renate und meiner Oma Waltraud.*

ACKNOWLEDGMENTS

First and foremost I want to thank my advisor Dr. Kenneth Stanley, for being a constant source of inspiration. I appreciate all his contributions of time and ideas to make my Ph.D. experience stimulating and productive.

Thanks to my dissertation committee members Dr. Charles Hughes, Dr. Gita Sukthankar and Dr. Paul Wiegand for offering their advice and donating their time and suggestions. Special thanks to Dr. Charles Hughes for serving as my initial advisor and providing me with funding during the first year of my Ph.D. career.

Thanks to past and present members of the Evolutionary Complexity Research Group (EPlex) at University of Central Florida (<http://eplex.cs.ucf.edu>) for their valuable feedback on my various papers, projects, and presentations. Especially, Amy Hoover, Joel Lehman, Dr. David D'Ambrosio, Dr. Charles E. Bailey, Dr. Ben Jones, Dr. Phillip Verbancsics, Paul Szerlip, and Brian Woolley.

Thanks also to DARPA and ARO for funding the majority of this research through DARPA grants HR0011- 08-1-0020, HR0011-09-1-0045 and N11AP20003 (Computer Science Study Group Phases 1, 2, and 3), and US Army Research Office grant Award No. W911NF-11-1-0489. This dissertation does not necessarily reflect the position or policy of the government, and no official endorsement should be inferred.

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Contributions	5
1.2 Outline	7
CHAPTER 2 BACKGROUND	8
2.1 Artificial Neural Networks	8
2.2 Neuroevolution	10
2.3 Neuroevolution of Augmenting Topologies (NEAT)	12
2.4 Generative and Developmental Systems (GDS)	15
2.4.1 Compositional Pattern Producing Networks (CPPNs)	15
2.4.2 HyperNEAT	17
2.5 Evolution and Learning	21
2.5.1 Evolution of Adaptive ANNs	22
2.5.2 Evolution of Neuromodulated Plasticity	24
2.6 Novelty Search	27
CHAPTER 3 ESTABLISHING THE POTENTIAL ADVANTAGE OF EVOLVING PLASTIC NEURAL NETWORKS WITH NOVELTY SEARCH	30

3.1	The Deceptive Trap of Learning to Learn	31
3.2	Extending NEAT to Evolve Neuromodulated ANNs	32
3.3	The Discrete T-Maze Domain	33
3.4	T-Maze Experiment	36
3.4.1	Measuring Novelty in the Single T-Maze	38
3.4.2	Generalization Performance	41
3.5	Single T-Maze Results	42
3.6	Analysis of Deception in the Single T-Maze	46
3.7	Additional Experiments	56
3.7.1	Discrete Double T-Maze	57
3.7.2	Foraging Bee	58
3.8	Implications of Combining Novelty Search with Plasticity	61
CHAPTER 4 APPROACH PART I: ADAPTIVE HYPERNEAT		66
4.1	Indirectly Encoding Local Learning Rules	66
4.2	Discrete T-Maze Experiment	69
4.3	Discrete T-Maze Results	72
4.4	Adaptive HyperNEAT Implications	74
CHAPTER 5 APPROACH PART II: EVOLVABLE-SUBSTRATE HYPERNEAT		77
5.1	Implicit Information in the Hypercube	78
5.2	Quadtree Information Extraction	80
5.3	ES-HyperNEAT Algorithm	85

5.4	Key Hypotheses	88
5.5	Experiment 1: Dual Task	91
5.5.1	Experimental Setup	93
5.5.2	Dual Task Results	95
5.6	Experiment 2: Maze Navigation	96
5.6.1	Maze Navigation Results	98
5.6.2	Example Solution Lineage	100
5.6.3	Evolvability Analysis	103
5.7	Experiment 3: Left & Right Retina Problem	105
5.7.1	Link Expression Output	106
5.7.2	ES-HyperNEAT Geometry Seeding	108
5.7.3	Retina Problem Setup	111
5.7.4	Results	113
5.8	ES-HyperNEAT Implications	115
5.8.1	Dictating Node Locations	117
5.8.2	Incrementally Building on Stepping Stones	118
5.8.3	Geometry Seeding	120
CHAPTER 6 UNIFIED APPROACH: ADAPTIVE ES-HYPERNEAT		122
6.1	Evolving Plasticity and Neural Geometry	122
6.2	Continuous T-Maze Domain	124
6.3	Continues T-Maze Results	127

6.4 Adaptive ES-HyperNEAT Implications	129
CHAPTER 7 TOWARDS AN ADAPTIVE NEUROCONTROLLER FOR LEGGED	
LOCOMOTION	131
7.1 Static Quadruped Neural Controller	133
7.2 Flexible Neural Controller	136
7.3 Experimental Setup	138
7.4 Quadruped Results	139
7.4.1 Interpolation Performance	140
7.4.2 Relationship Between Morphology and Control	142
7.5 Implications of Flexible Controllers	143
CHAPTER 8 DISCUSSION AND FUTURE WORK	
8.1 Towards More Brain-Like ANNs	146
8.2 HyperNEAT Extensions	147
8.3 Future Directions	149
CHAPTER 9 CONCLUSION	
9.1 Contributions	152
9.2 Conclusion	154
APPENDIX: PARAMETERS AND PSEUDOCODE	
LIST OF REFERENCES	178

LIST OF FIGURES

Figure 2.1	CPPN Encoding	17
Figure 2.2	Hypercube-based Geometric Connectivity Pattern Interpretation . . .	19
Figure 2.3	Neuromodulated Plasticity	25
Figure 3.1	The Discrete T-Maze	34
Figure 3.2	T-Maze ANN Topology	37
Figure 3.3	The T-Maze Novelty Metric	39
Figure 3.4	Three T-Maze Sample Behaviors	40
Figure 3.5	Comparing Generalization of Novelty Search and Fitness-based Search	43
Figure 3.6	Average Evaluations to Solution for Novelty Search and Fitness-based Search	44
Figure 3.7	Novelty Search Archive and Fitness Champions	47
Figure 3.8	Combined Sammon’s Mapping	51
Figure 3.9	Sammon’s Mapping of Novelty and Fitness-based Search at Different Stages of Evolution	54
Figure 3.10	Distribution of Fitness for Novelty and Fitness-based Search for a Typ- ical Run	55
Figure 3.11	The Double T-Maze	57
Figure 3.12	Comparing Novelty Search to Fitness-based Search in the Bee Domain	59
Figure 4.1	Adaptive HyperNEAT	68

Figure 4.2 T-Maze and Substrate Configuration	70
Figure 4.3 Nonlinear Reward Color Encoding	71
Figure 4.4 T-Maze Training Performance	73
Figure 4.5 Discovered Learning Rules of an ANN Solution Created by the Iterated Model and the Underlying CPPN	74
Figure 5.1 Quadtree Information Extraction Example	82
Figure 5.2 Example Connection Selection	84
Figure 5.3 The ES-HyperNEAT Algorithm	86
Figure 5.4 The Dual Task	91
Figure 5.5 Substrate Configuration and Sensor Layout	93
Figure 5.6 FS-HyperNEAT Hidden Node Layouts	94
Figure 5.7 Average Dual Task Performance	96
Figure 5.8 Maze Navigation Domain	97
Figure 5.9 Average Performance and Champion Complexity	99
Figure 5.10 ANN Milestones and Underlying CPPNs Together With the Agent's behavior From a Single Maze Solution Lineage	102
Figure 5.11 Comparing the Evolvability of Fixed-Substrate and Evolvable-Substrate HyperNEAT	104
Figure 5.12 The Retina Problem	106
Figure 5.13 X-locality and Geometry Seeding	109
Figure 5.14 Retina Problem Substrate Configuration	110

Figure 5.15 Average Performance in Retina Domain	114
Figure 5.16 Example Connectivity Patterns from ES-HyperNEAT-LEO with Ge- ometry Seeding in a Modular Domain	116
Figure 6.1 Adaptive ES-HyperNEAT CPPN	123
Figure 6.2 Continues T-Maze Domain and Substrate Configuration	126
Figure 6.3 Average Performance in Continues T-Maze	128
Figure 6.4 Example Solution ANN Created by Adaptive ES-HyperNEAT and its Underlying CPPN	128
Figure 7.1 Quadruped Training Morphologies	133
Figure 7.2 Distributed Substrate Architecture and CPPN	134
Figure 7.3 Motor Neuron Dynamics	141
Figure 7.4 Best Interpolation Performance	141
Figure 7.5 Morphology-Dependent Change	142

LIST OF TABLES

Table A.1 Parameter Settings in HyperNEAT Experiments	161
---	-----

CHAPTER 1

INTRODUCTION

The human brain is the most complex system known to exist (Kandel et al., 1991; Zigmond et al., 1999). It is composed of an astronomically high number of neurons and connections organized in precise, intricate motifs that repeat throughout it, often with variation on a theme, such as cortical columns (Sporns, 2002). Additionally, the brain is not a static entity but instead adaptive and changing throughout life, enabling us to learn and modify our behavior based on past experience. These characteristics have so far eluded attempts to create artificial neural networks (ANNs) with the same capabilities.

One methodology that has shown promise in autonomously generating ANNs for complex control problems is *neuroevolution*, i.e. evolving ANNs through evolutionary algorithms (EAs) (Floreano et al., 2008; Reil and Husbands, 2002; Stanley et al., 2005; Stanley and Miikkulainen, 2002; Yao, 1999). To this end, as such algorithms are asked to evolve increasingly large and complex structures, interest has increased in recent years in indirect neural network encodings, wherein the description of the solution is compressed such that information can be reused (Bongard, 2002; Gauci and Stanley, 2010; Hornby and Pollack, 2002; Stanley, 2007; Stanley and Miikkulainen, 2003). Such compression allows the final solution to contain more components than its description. Nevertheless, neuroevolution has historically produced networks with orders of magnitude fewer neurons and significantly less organization and regularity than natural brains (Stanley and Miikkulainen, 2004; Yao, 1999).

While past approaches to neuroevolution generally concentrated on deciding which node is connected to which (i.e. neural *topology*) (Floreano et al., 2008; Stanley and Miikkulainen, 2004; Yao, 1999), the recently introduced Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) method (D’Ambrosio and Stanley, 2007; Gauci and Stanley, 2008; Stanley et al., 2009) provided a new perspective on evolving ANNs by showing that the pattern of weights across the connectivity of an ANN can be generated as a function of its geometry. HyperNEAT employs an indirect encoding called compositional pattern producing networks (CPPNs; Stanley 2007), which can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation. In effect, the CPPN in HyperNEAT paints a pattern within a four-dimensional hypercube that is interpreted as the isomorphic connectivity pattern.

HyperNEAT exposed the fact that neuroevolution benefits from neurons that exist at *locations* within the space of the brain and that by placing neurons at locations, evolution can exploit *topography* (as opposed to just topology), which makes it possible to correlate the geometry of sensors with the geometry of the brain. While lacking in many ANNs, such geometry is a critical facet of natural brains that is responsible for e.g. topographic maps and modular organization across space (Sporns, 2002). This insight allowed large ANNs with regularities in connectivity to evolve for high-dimensional problems (Clune et al., 2011; Gauci and Stanley, 2008, 2010; Stanley et al., 2009). In a step towards more brain-like neurocontrollers, this dissertation presents two extensions to HyperNEAT, which extend its reach to more complex domains.

In contrast with real brains HyperNEAT is so far only able to produce *static* networks, which means that the synaptic connections in the ANN do not change their strength during the lifetime of the network. While some tasks do not require the network to change its behavior, many domains would benefit from *online adaptation*. In other words, whereas evolution produces phylogenetic adaptation, learning gives the individual the possibility to react much faster to environmental changes by modifying its behavior during its lifetime. For example, a robot that is physically damaged should be able to adapt to its new circumstances without the need to re-evolve its neurocontroller. One way that agents controlled by ANNs can evolve the ability to adapt over their lifetime is by encoding *local learning rules* in the genome that determine how their synaptic connection strengths should change in response to changing activation levels in the neurons they connect (Floreano and Urzelai, 2000; Niv et al., 2002; Soltoggio et al., 2008). This approach resembles the way organisms in nature, which possess plastic nervous systems, cope with changing and unpredictable environments. Although demonstrations of this approach have suggested the promise of evolving adaptive ANNs, a significant problem is that local learning rules for every connection in the network must be discovered separately (Floreano and Urzelai, 2000).

The first HyperNEAT extension introduced in this dissertation, which is called *adaptive HyperNEAT*, addresses this problem and shows that not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of *local neural learning rules*. The idea that learning rules can be distributed in a geometric pat-

tern is new to neuroevolution but reflects the intuition that synaptic plasticity in biological brains is not encoded in DNA separately for every synapse in the brain.

The second HyperNEAT extension addresses the significant limitation of the original HyperNEAT approach that the *positions* of the nodes must be decided a priori by the user. In other words, in the original HyperNEAT, the user must literally place nodes (i.e. neurons) at locations within a two-dimensional or three-dimensional space called the substrate. The new approach, called *evolvable-substrate HyperNEAT* (ES-HyperNEAT), shows that the placement and density of the hidden nodes can in fact be determined solely based on implicit information in an infinite-resolution pattern of weights generated by HyperNEAT. That way, the user no longer needs to decide their positions. The novel insight is that a representation that encodes the pattern of connectivity across a network (such as in HyperNEAT) automatically contains *implicit* clues on where the nodes should be placed to best capture the information stored in the connectivity pattern. In other words, there is no need for any new information or any new representational structure beyond the very same CPPN that already encodes network connectivity in the original HyperNEAT. This approach not only can evolve the location of every neuron in the network, but also can represent regions of varying density, which means resolution can increase holistically over evolution.

Based on the insights behind adaptive HyperNEAT and ES-HyperNEAT, the two approaches are then combined to encode the geometry, density, and plasticity of an evolving ANN. This

new method is called *adaptive ES-HyperNEAT*, which significantly expands the scope of neural structures that evolution can discover.

In addition to adaptive ES-HyperNEAT, this dissertation presents work in the major application domain of legged locomotion. First steps towards the general goal of a plastic neurocontroller that can adapt to a variety of different robot body proportions are taken by showing that HyperNEAT can learn the relationship between the morphology and control of different quadruped robots. Highlighting the unique potential of such an approach, the evolved neurocontrollers can interpolate to never-seen intermediate morphologies *without any further training*.

The idea that we are beginning to be able to reproduce some of the phenomena produced through natural evolution at a *high level of abstraction* is important for the field of AI because the evolution of brains ultimately produced the seat of intelligence in nature.

1.1 Contributions

The research hypothesis of this dissertation is that evolving more brain-like ANNs will allow neuroevolution to more easily solve complex control problems and adaptive tasks than current neuroevolution technology. This hypothesis is supported by the following contributions:

1. The advantages of evolving plastic neural networks with the recently introduced novelty search method (Lehman and Stanley, 2008, 2011b) are established (Chapter 3). Highlighting this advantage is important to this dissertation because of its later focus on evolving such plastic network.
2. Adaptive HyperNEAT extends HyperNEAT to allow not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of arbitrary learning rules (Chapter 4).
3. ES-HyperNEAT extends HyperNEAT to determine the placement and density of the hidden nodes automatically based on implicit information in an infinite-resolution pattern of weights (Chapter 5).
4. The combined approach, adaptive ES-HyperNEAT, unifies for the first time the ability to indirectly encode connectivity through geometry, simultaneously encode the density and placement of nodes in space, and generate patterns of heterogeneous plasticity (Chapter 6).
5. A step towards the general goal of adaptive controllers for locomotion is taken by showing that HyperNEAT allows learning the relationship between the morphology and control of different quadruped robots (Chapter 7).

1.2 Outline

The next chapter begins with relevant background on artificial neural network, neuroevolution, the evolution of adaptive ANNs, generative and developmental systems, and a review of HyperNEAT. Chapter 3 establishes the advantages of evolving plastic neural network with novelty search. Next, adaptive HyperNEAT is introduced in Chapter 4, followed by the evolvable-substrate extension to HyperNEAT in Chapter 5. The unified approach, adaptive ES-HyperNEAT, is introduced in Chapter 6. Chapter 7 then presents initial experiments towards an adaptive neurocontroller for legged locomotion. Finally, a discussion and conclusions are in Chapter 8 and 9.

CHAPTER 2

BACKGROUND

This chapter reviews several areas relevant to ES-HyperNEAT and adaptive HyperNEAT, including the basics of neural networks, the field of neuroevolution, the evolution of adaptive ANNs, the field of generative and developmental systems, and the original HyperNEAT.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models based on the architecture of the human brain. The fundamental processing units in neural networks are neurons, which are connected by weighted links to each other to form the full network. A neuron (also called a node) in an ANN is usually classified as one of three types: input, hidden or output. While input nodes receives information from the external world, the output nodes of the network control the behavior of the system (e.g. a robot) containing the ANN. The neurons between the input and output nodes, where most processing takes place, are the hidden nodes.

Each neuron j in a typical neural network computes the weighted sum of its incoming signals passed through an activation function Φ that squashes the activation into a certain range

(typically between 0 and 1 or -1 to 1) and calculates output y_j given an input vector \mathbf{x} :

$$y_j = \Phi \left(\sum_{i=1} (w_{ij} x_i) \right), \quad (2.1)$$

where w_{ij} is the connection weight between neuron i and neuron j . One of the most common activation functions is the sigmoid, or logistic function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (2.2)$$

All experiments in this dissertation, except the legged locomotion experiment described in Chapter 7, employ the neural model described above. While this model is sufficient to generate neural controllers for a broad range of problems, it does not allow a neuron to display varied temporal behavior. For example, complex oscillating patterns, like the non-linear dynamics found in natural gaits, seem to rely on more realistic models of neural activation (Reil and Husbands, 2002; McHale and Husbands, 2004).

A model with this capability is the *continuous-time recurrent neural network* (CTRNN; Yamauchi and Beer 1994). The neurons in CTRNNs are called *leaky integrator neurons*. Such neurons integrate incoming activation over time (instead of all at once as in the previous model) and also gradually leak internal activation (i.e. the neuron’s activation level slowly decreases to zero without any input). The rate of integration and leak is controlled by a *time constant* τ , which allows different neurons to exhibit different temporal dynamics. A discrete time update rule (Blynell and Floreano, 2002) of the internal activation level γ takes

the following form

$$\gamma_j(n+1) = \gamma_j(n) + \frac{\Delta t}{\tau_j} \left(-\gamma_j(n) + \sum_{i=1}^N w_{ij} A_i(n) + \sum_{k=1}^S w_{kj} I_k \right), \quad (2.3)$$

where N is the total numbers of neurons, S is the number of inputs, I_k is the sensory activation, and A_i is the incoming activation from neuron i . The legged controllers evolved in Chapter 7 are controlled by CTRNNs because they require sophisticated oscillatory patterns. The next section describes a method for training neural networks called neuroevolution.

2.2 Neuroevolution

Neuroevolution (NE) focuses on training ANNs through evolutionary algorithms. It can serve as an alternative to value function reinforcement learning (RL) methods (Sutton and Barto, 1998; Kaelbling et al., 1996) and hand designed cortical simulations (Lytton and Sejnowski, 1991; Izhikevich, 2004), and has shown promise in a variety of difficult control tasks (Floreano et al., 2008; Reil and Husbands, 2002; Stanley et al., 2005; Stanley and Miikkulainen, 2002; Yao, 1999).

A typical evolutionary algorithm works by first evaluating a set (population) of individuals that are assigned a score (fitness) based on their performance in a given task. In the next

step, the new population is created by probabilistically choosing the individuals with higher fitness. The individuals of this next generation are created by probabilistically applying mutation and crossover operations to the parent individuals. This generational process is iteratively repeated until an individual with a high enough fitness value is found.

The *genotype* in NE is a compact representation of specific features of a network that is translated into the final ANN *phenotype*. The advantage of neuroevolution is that it allows the evolution of any kind of ANN, including recurrent and adaptive networks (Section 2.5.1). Early NE research focused on only evolving the synaptic weights, while the topology of the ANN remained fixed during evolution (Gomez and Miikkulainen, 1999; Saravanan and Fogel, 1995). While this approach allowed the optimization of the network topology by a human expert, it is not trivial to determine the right topology a priori. Therefore interest in recent years has increased in methods that can discover the right neural architecture and weights on their own (Angeline et al., 1994; Gruau et al., 1996; Yao, 1999).

The next section reviews such a method, called NEAT, which is the foundation of the HyperNEAT method that is itself extended in this dissertation.

2.3 Neuroevolution of Augmenting Topologies (NEAT)

The NEAT method was originally developed to evolve ANNs to solve difficult control and sequential decision tasks and has proven successful in a wide diversity of domains (Aalto-nen et al., 2009; Stanley et al., 2003, 2005; Stanley and Miikkulainen, 2002; Taylor et al., 2006; Whiteson and Stone, 2006). Evolved ANNs control agents that select actions based on their sensory inputs. NEAT is unlike many previous NE methods, which traditionally evolve either fixed-topology networks (Gomez and Miikkulainen, 1999; Saravanan and Fogel, 1995), or arbitrary random-topology networks (Angeline et al., 1994; Gruau et al., 1996; Yao, 1999). Instead, NEAT begins evolution with a population of small, simple networks and complexifies the network topology into diverse species over generations, leading to increasingly sophisticated behavior. A similar process of gradually adding new genes has been confirmed in natural evolution (Martin, 1999; Watson et al., 1987) and shown to improve adaptation in a few prior evolutionary (Watson et al., 1987) and neuroevolutionary (Harvey, 1993) approaches. However, a key feature that distinguishes NEAT from prior work in complexification is its unique approach to maintaining a healthy diversity of complexifying structures simultaneously, as this section reviews. Complete descriptions of the NEAT method, including experiments confirming the contributions of its components, are available in Stanley and Miikkulainen (2002, 2004) and Stanley et al. (2005).

The NEAT method is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is which.

Otherwise, it is not clear in later generations which individual is compatible with which in a population of diverse structures, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique historical marking to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover among diverse topologies without the need for expensive topological analysis.

Second, historical markings make it possible for the system to divide the population into species based on how similar they are topologically. That way, individuals compete primarily within their own niches instead of with the population at large. Because adding new structure is often initially disadvantageous, this separation means that unique topological innovations are protected and therefore have time to optimize their structure before competing with other niches in the population. The distance δ between two network encodings can be measured as a linear combination of the number of excess (E) and disjoint (D) genes, as well as the average weight differences of matching genes (\overline{W}), where *excess* genes are those that arise in the lineage of one parent at a time later than all the genes in the other parent and *disjoint* genes are any other genes in the lineage of one parent but not the other one (Stanley and Miikkulainen, 2002, 2004):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}. \quad (2.4)$$

The coefficients c_1 , c_2 , and c_3 adjust the importance of the three factors, and the factor N , the number of genes in the larger genome, normalizes for genome size (N is normally set to one unless both genomes are excessively large; accordingly, it is set to one in the experiments in this dissertation). Genomes are tested one at a time; if a genome’s distance to a representative member of the species is less than δ_t , a compatibility threshold, the genome is placed into this species.

Third, many systems that evolve network topologies and weights begin evolution with a population of random topologies (Gruau et al., 1996; Yao, 1999). In contrast, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. Because of speciation, novel topologies gradually accumulate over evolution, thereby allowing diverse and complex phenotype patterns to be represented. No limit is placed on the size to which topologies can grow. New structures are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In effect, then, NEAT searches for a compact, appropriate topology by incrementally increasing the complexity of existing structure.

The next section reviews generative and developmental systems (GDS), focusing on compositional pattern producing networks (CPPNs) and the HyperNEAT approach, which will be extended in this dissertation.

2.4 Generative and Developmental Systems (GDS)

In *direct encodings* like NEAT, each part of the solution’s representation maps to a single piece of structure in the final solution (Floreano et al., 2008; Yao, 1999). The significant disadvantage of this approach is that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. Thus this dissertation employs an *indirect* encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself. Indirect encodings are powerful because they allow solutions to be represented as a *pattern* of parameters, rather than requiring each parameter to be represented individually (Bentley and Kumar, 1999; Bongard, 2002; Gauci and Stanley, 2010; Hornby and Pollack, 2002; Stanley, 2007; Stanley and Miikkulainen, 2003). The next section reviews one such indirect encodings in more detail, called compositional pattern producing networks.

2.4.1 Compositional Pattern Producing Networks (CPPNs)

Recently, NEAT has been extended to evolve a high-level developmental abstraction called compositional pattern producing networks (CPPNs) (Stanley, 2007). The idea behind CPPNs is that patterns in nature can be described at a high level as compositions of functions,

wherein each function in the composition represents a stage in development. CPPNs are similar to ANNs, but they rely on more than one activation function (each representing a common regularity).

The indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation (Secretan et al., 2008, 2011; Stanley, 2007). For example, simply by including a Gaussian function, which is symmetric, the output pattern can become symmetric. A periodic function such as sine creates segmentation through repetition. Most importantly, *repetition with variation* (e.g. such as the fingers of the human hand) is easily discovered by combining regular coordinate frames (e.g. sine and Gaussian) with irregular ones (e.g. the asymmetric x-axis). For example, a function that takes as input the sum of a symmetric function and an asymmetric function outputs a pattern with imperfect symmetry. In this way, CPPNs produce regular patterns with subtle variations. The potential for CPPNs to represent patterns with motifs reminiscent of patterns in natural organisms has been demonstrated in several studies (Secretan et al., 2008, 2011; Stanley, 2007).

Specifically, CPPNs produce a phenotype that is a function of n dimensions, where n is the number of dimensions in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 2.1 shows how a two-dimensional phenotype can be generated by a function of two parameters that is represented by a network of composed functions. Because CPPNs are a superset of traditional

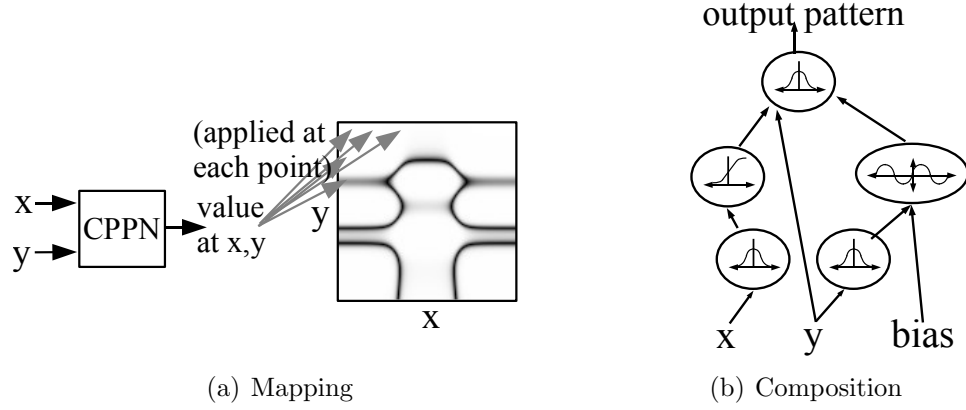


Figure 2.1: **CPPN Encoding.** (a) The function f takes arguments x and y , which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of f , the result is a spatial pattern, which can be viewed as a phenotype whose genotype is f . (b) The CPPN is a graph that determines which functions are connected. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection.

ANNs, which can approximate any function (Cybenko, 1989), CPPNs are also universal function approximators. Thus a CPPN can encode any pattern within its n -dimensional space. The next section reviews the HyperNEAT extension to NEAT that is itself extended in this dissertation.

2.4.2 HyperNEAT

HyperNEAT, reviewed in this section, is an indirect encoding extension of NEAT that is proven in a number of challenging domains that require discovering regularities (Clune et al.,

2009; Drchal et al., 2009; Gauci and Stanley, 2008, 2010; Stanley et al., 2009). For a full description of HyperNEAT see Stanley et al. (2009) and Gauci and Stanley (2010).

The main idea in HyperNEAT is to extend CPPNs, which encode spatial patterns, to also represent connectivity patterns (Clune et al., 2009; Gauci and Stanley, 2008, 2010; Stanley et al., 2009). That way, NEAT can evolve CPPNs that represent large-scale ANNs with their own symmetries and regularities. The key insight is that $2n$ -dimensional spatial patterns are isomorphic to connectivity patterns in n dimensions, i.e. in which the coordinate of each endpoint is specified by n parameters. Consider a CPPN that takes four inputs labeled $x1$; $y1$; $x2$; and $y2$; this point in four-dimensional space also denotes the connection between the two-dimensional points $(x1; y1)$ and $(x2; y2)$, and the output of the CPPN for that input thereby represents the weight of that connection (figure 2.2). By querying every possible connection among a set of points in this manner, a CPPN can produce a neural network, wherein each queried point is a neuron position. Because the connections are produced by a function of their endpoints, the final structure is produced with knowledge of its geometry. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as an isomorphic connectivity pattern, which explains the origin of the name Hypercube-based NEAT (HyperNEAT). Connectivity patterns produced by a CPPN in this way are called substrates so that they can be verbally distinguished from the CPPN itself, which has its own internal topology.

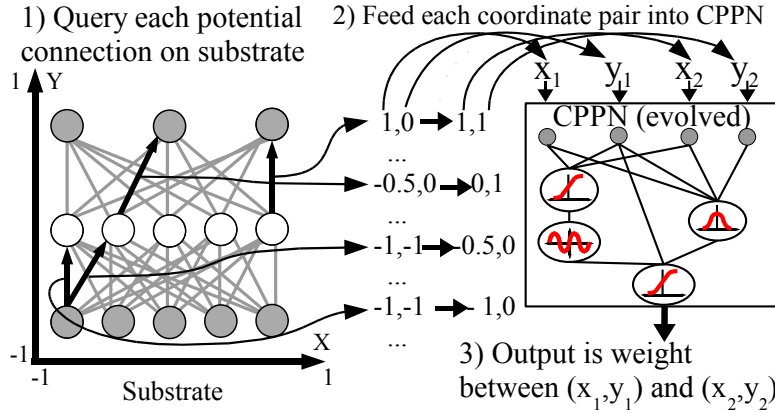


Figure 2.2: **Hypercube-based Geometric Connectivity Pattern Interpretation.** A collection of nodes, called the *substrate*, is assigned coordinates that range from -1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) Internally, the CPPN (which is evolved) is a graph that determines which activation functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connections in space.

Each queried point in the substrate is a node in an ANN. In traditional implementations of HyperNEAT the experimenter defines both the location and role (i.e. hidden, input, or output) of each such node. As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the task (Clune et al., 2009; Drchal et al., 2009; Gauci and Stanley, 2008; Stanley et al., 2009). That way, the connectivity of the substrate is a function of the task structure.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot. Outputs for moving left or right can also be placed in the same order, allowing HyperNEAT to understand from the outset the correlation of sensors to effectors. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry) of a problem that are invisible to traditional encodings. Yet a problem that has endured with HyperNEAT is that the experimenter is left to decide how many hidden nodes there should be and where to place them. That is, although the CPPN determines how to connect nodes in a geometric space, it does not specify where the nodes should be. Before ES-HyperNEAT, introduced in this dissertation, no automated solution to this problem had yet been introduced, in part because it seems to require some kind of additional structure or apparatus beyond the CPPN representation.

In answer to this challenge, I present the evolvable-substrate extension to HyperNEAT in Chapter 5 of this dissertation, in which the placement and density of the hidden nodes do

not need to be set a priori. In fact, they are completely determined by implicit information in the CPPN itself.

2.5 Evolution and Learning

Unlike the synaptic connections in living brains, the synaptic connections in ANNs produced by NE are normally static, which may limit the adaptive dynamics the network can display during its lifetime (Blynel and Floreano, 2002). While some tasks do not require the network to change its behavior, many domains would benefit from *online adaptation*. In other words, whereas evolution produces phylogenetic adaptation, learning gives the individual the possibility to react much faster to environmental changes by modifying its behavior during its lifetime. For example, a robot that is physically damaged or altered should be able to adapt to its new circumstances without the need to re-evolve its neurocontroller. In this way, when the environment changes from what was encountered during evolution, adapting online is often necessary to maintain performance. One contribution of this dissertation is to add this capability to HyperNEAT.

There is much evidence that evolution and learning are both integral to the success of biological evolution (Mayley, 1997; Nolfi and Floreano, 1999) and that lifetime learning itself can help to guide evolution to higher fitness (Hinton and Nowlan, 1987).

Thus NE can benefit from combining these complementary forms of adaptation by evolving ANNs with synaptic plasticity driven by local learning rules (Baxter, 1992; Floreano and Urzelai, 2000; Stanley et al., 2003). Synaptic plasticity allows the network to change its internal connection weights based on experience during its lifetime. It also resembles the way organisms in nature, which possess plastic nervous systems, cope with changing and unpredictable environments (Floreano and Urzelai, 2000; Niv et al., 2002; Soltoggio et al., 2008).

The next section reviews the evolution of adaptive ANNs and details a model for neuromodulated plasticity, which is a recent advance in plastic neurocontrollers.

2.5.1 Evolution of Adaptive ANNs

Researchers have been evolving adaptive ANNs for more than fifteen years. Early work often focused on combining the built-in adaptive capabilities of backpropagation (Rumelhart et al., 1986) with NE. For example, Nolfi and Parisi (1993, 1996) evolved self-teaching networks that train a motor control network through backpropagation from the outputs of a teaching subnetwork. In separate work, they evolved a network that learns through backpropagation to predict what it would see after moving around in its environment (Nolfi et al., 1994). Learning to predict the next state during the network’s lifetime was shown to enhance performance in a foraging task. Interestingly, Chalmers (1990) evolved a global learning

rule (i.e. a rule that applies to every connection) and discovered that the evolved rule is similar to the well-known delta rule used in backpropagation. Furthermore, McQuesten and Miikkulainen (1997) showed that NE can benefit from parent networks teaching their offspring through backpropagation.

Baxter (1992) performed early work on evolving networks with synaptic plasticity driven by local learning rules, setting the stage for NE of adaptive ANNs. He evolved a very simple network that could learn boolean functions of one value. Each connection had a rule for changing its weight to one of two possible values. Baxter’s contribution was mainly to show that local learning rules are sufficient to evolve an adaptive network. Floreano and Urzelai (2000) later showed that the evolution of local (node-based) synaptic plasticity parameters produces networks that can solve complex problems better than networks with fixed-weights.

In Floreano and Urzelai’s experiment, an adaptive network and a fully-connected network were evolved to turn on a light by moving to a switch. After the light turned on, the networks had to move onto a gray square. Adaptive networks were compared to fixed-weight networks. Each connection in the adaptive network included a learning rule and a learning rate. The fixed-weight network only encoded static connection weights. The sequence of two actions proved difficult to learn for the fixed-weight network because the network could not adapt to the sudden change in goals after the light was switched on. Fixed-weight networks tended to circle around the environment, slightly attracted by both the light switch and the gray square. Adaptive networks, on the other hand, completely changed their trajectories after

turning on the light, reconfiguring their internal weights to tackle the problem of finding the gray square. This landmark result established the promise of evolving adaptive ANNs and that in fact adaptive networks can sometimes evolve faster than static networks. The local learning rules in the evolved networks facilitated the policy transition from one task to the other.

Adaptive ANNs have also been successfully evolved to simulate robots in a dangerous foraging domain (Stanley et al., 2003). Although this work also showed that recurrent fixed-weight networks can be more effective and reliable than adaptive Hebbian controllers in some domains, more recent studies (Niv et al., 2002; Soltoggio et al., 2008, 2007) suggest that both network types reach their limits when more elaborate forms of learning are needed. For example, classical conditioning seems to require mechanisms that are not present in most current network models. To expand to such domains, the adaptive ES-HyperNEAT approach presented in Chapter 6 will control synaptic plasticity through *neuromodulation*, which is explained in the next section.

2.5.2 Evolution of Neuromodulated Plasticity

In the plastic ANNs presented in the previous section (e.g. Floreano and Urzelai 2000; Stanley et al. 2003), the internal synaptic connection strengths change following a Hebbian learning rule that modifies synaptic weights based on pre- and postsynaptic neuron activity. The

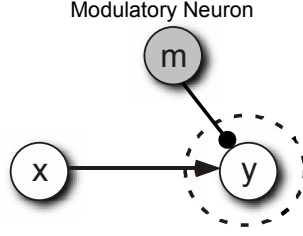


Figure 2.3: **Neuromodulated Plasticity.** The weight of the connection between standard neurons n_1 and n_2 is modified by a Hebbian rule. Modulatory neuron m determines the magnitude of the weight change.

generalized Hebbian plasticity rule (Niv et al., 2002) takes the following form:

$$\Delta w = \eta \cdot [Axy + Bx + Cy + D], \quad (2.5)$$

where η is the learning rate, x and y are the activation levels of the presynaptic and postsynaptic neurons, and A – D are the correlation term, presynaptic term, postsynaptic term, and constant, respectively.

In a *neuromodulated* network, a special neuromodulatory neuron can change the degree of potential plasticity between two standard neurons in response to their activation levels (figure 2.3). In addition to its standard activation value a_j , each neuron j in such a network also computes its modulatory activation m_j :

$$a_j = \sum_{i \in Std} w_{ij} \cdot o_i, \quad (2.6)$$

$$m_j = \sum_{i \in Mod} w_{ij} \cdot o_i, \quad (2.7)$$

where w_{ij} is the connection strength between presynaptic neuron i and postsynaptic neuron j and o_i is calculated as $o_i(a_i) = \tanh(a_i/2)$. The weight between neurons i and j then changes following the m_i -modulated plasticity rule

$$\Delta w_{ij} = \tanh(m_j/2) \cdot \eta \cdot [A o_i o_j + B o_i + C o_j + D]. \quad (2.8)$$

The benefit of adding modulation is that it allows the ANN to change the level of plasticity on specific neurons at specific times. That is, it becomes possible to decide when learning should *stop* and when it should *start*. This property seems to play a critical role in regulating learning behavior in animals (Carew et al., 1981) and neuromodulated networks have a clear advantage in more complex dynamic, reward-based scenarios: Soltoggio et al. (2008) showed that networks with neuromodulated plasticity significantly outperform both fixed-weight and traditional adaptive ANNs without neuromodulation in a double T-Maze learning domain, and display nearly optimal learning performance.

However, evolving adaptive controllers for more complicated tasks has proven difficult in part because learning to learn can be highly deceptive: Reaching a mediocre fitness through *non-adaptive* behavior is often relatively easy, but any further improvement requires an improbable leap to sophisticated adaptive behavior. Therefore the next section reviews a method called *novelty search* (Lehman and Stanley, 2008, 2011b), which abandons objective-based fitness and instead simply searches *only* for novel behavior, which avoids deception

entirely. The advantages of training plastic ANNs with novelty search in particular are then established in Chapter 3.

2.6 Novelty Search

The problem with the objective fitness function in evolutionary computation is that it does not necessarily reward the intermediate stepping stones that lead to the objective. The more ambitious the objective, the harder it is to identify *a priori* these stepping stones.

A potential solution to this problem is novelty search, which is a recent method for avoiding deception based on the radical idea of ignoring the objective (Lehman and Stanley, 2008, 2011b). The idea is to identify novelty as a proxy for stepping stones. That is, instead of searching for a final objective, the learning method is rewarded for finding any behavior whose functionality is significantly different from what has been discovered before. Thus, instead of an objective function, search employs a *novelty metric*. That way, no attempt is made to measure overall progress. In effect, such a process gradually accumulates novel behaviors.

Although it is counterintuitive, novelty search was actually *more effective* at finding the objective than a traditional objective-based fitness function in a deceptive navigation domain that requires an agent to navigate through a maze to reach a specific goal location

(Lehman and Stanley, 2008; Mouret, 2009), in evolving biped locomotion (Lehman and Stanley, 2011b), and in evolving a program for an artificial ant benchmark task (Lehman and Stanley, 2010a).

To implement novelty search in an evolutionary algorithm, the fitness function has to be replaced by a *novelty metric*, which measures the difference between the *behaviors* of two individuals. Once the novelty metric is defined, the novelty of a new behavior can be computed by comparing it to the individuals of the current population and an archive of individuals whose behaviors were highly novel when they first appeared. The farther away the behavior of a new individual is from the rest of the population in the space of unique behaviors the higher should be its novelty. Thus novelty at point x in the behavior space can be measured based on the *spareness* ρ of behaviors at that point in the novelty space and calculated as the average distance to its k -nearest neighbors

$$\rho(x) = \frac{1}{k} \sum_{i=1}^k \text{dist}(x, \mu_i), \quad (2.9)$$

where μ_i is the i th-nearest neighbor of x with respect to the novelty metric *dist*. If novelty is above some minimal threshold ρ_{min} , then the individual is entered into the permanent archive that characterizes the distribution of prior highly novel behaviors.

In summary, novelty search depends on the following four main concepts:

- Individuals' behaviors are *characterized* so that they can be compared.

- The novelty of an individual is computed with respect to observed *behaviors* of other individuals and not others' genotypes.
- Novelty search replaces the fitness function with a novelty metric that computes the sparseness at any point in the novelty space.
- An archive of past individuals is maintained whose behaviors were highly novel.

The next chapter describes the application of novelty search to the deceptive task of learning to learn.

CHAPTER 3

ESTABLISHING THE POTENTIAL ADVANTAGE OF EVOLVING PLASTIC NEURAL NETWORKS WITH NOVELTY SEARCH

This chapter analyzes the inherent deceptiveness in reinforcement learning problems and shows how novelty search (Lehman and Stanley, 2008, 2011b) can avoid such deception by exploiting intermediate stepping stones to guide its search. The experiments in this chapter, which are published in the *Adaptive Behavior* journal (Risi et al., 2010a), are a major extension of work first published at the Genetic and Evolutionary Computation Conference (GECCO 2009) (Risi et al., 2009). While these are not the main contributions of this dissertation, which are presented in later chapters, this chapter identifies novelty search as a new tool for evolving plastic neural networks (which are also a later focus) because such experiments are often thwarted by deception. For example, in Chapter 7 this technique helps to evolve solutions in a challenging legged locomotion domain that requires adaptation. Also importantly, to focus on the problem of deception in reinforcement learning domains, the ANNs in this chapter are evolved by NEAT, while the ANNs in the following chapters are evolved by more sophisticated indirect encoding extensions of HyperNEAT.

3.1 The Deceptive Trap of Learning to Learn

Deception in domains that require adaptation is particularly pathological for two primary reasons: (1) Reaching a mediocre fitness through *non-adaptive* behavior is often relatively easy, but any further improvement requires an improbable leap to sophisticated adaptive behavior, and (2) only sparse feedback on the acquisition of adaptive behavior is available from an objective-based performance measure. Because it is easier at first to improve fitness without evolving the ability to learn, objective functions may sometimes exploit domain-dependent static (i.e. non-adaptive) heuristics that can lead them further away from the adaptive solution in the genotypic search space, as analysis in this chapter will confirm. Because of the problem of deception in adaptive domains, prior experiments in evolving plastic ANNs have needed to be carefully designed to ensure that no non-adaptive heuristics exist that could potentially lead evolution prematurely astray. This awkward requirement has significantly limited the scope of domains amenable to adaptive evolution and stifled newcomers from entering the research area. To remedy this situation and open up the range of problems amenable to evolving adaptation, this chapter shows that the novelty search algorithm (Section 2.6; Lehman and Stanley 2008, 2011b), which abandons the traditional notion of objective-based fitness, circumvents the deception inherent in adaptive domains.

To demonstrate the potential of this approach, the following section compares novelty search to fitness-based evolution in a dynamic, reward-based single T-Maze scenario first studied in the context of NE by Blynel and Floreano (2003) and further investigated by Soltoggio et al.

(2008) to demonstrate the advantage of neuromodulated plasticity. In this scenario, the reward location is a variable factor in the environment that the agent must learn to exploit. Because the aim of this chapter is to show that novelty search solves particular difficult problems in the evolution of *plastic* networks and it has been shown that neuromodulation (Section 2.5.2) is critical to those domains (Soltoggio et al., 2008), all evolved ANNs employ this most effective form of plasticity. Additional experiments in the more complicated double T-Maze domain and a bee foraging task further test the hypothesis that novelty search can effectively overcome the deception inherent in many dynamic, reward-based scenarios.

3.2 Extending NEAT to Evolve Neuromodulated ANNs

Few modifications to the standard NEAT algorithm are required also to encode neuromodulated plasticity. NEAT’s genetic encoding is augmented with a new modulatory neuron type and each time a node is added through structural mutation, it is randomly assigned a standard or modulatory role. The neuromodulatory dynamics follow equations 2.6–2.8.

The coefficients of the generalized Hebbian learning rule used by all evolved neuromodulated networks in the T-Maze domain are $A = 0.0$, $B = 0.0$, $C = -0.38$, $D = 0.0$ and $\eta = -94.6$, resulting in the following m_i -modulated plasticity rule:

$$\Delta w_{ji} = \tanh(m_i/2) \cdot 35.95y. \quad (3.1)$$

These values worked well for a neuromodulated ANN in learning problems described by Soltoggio et al. (2008). Therefore, to isolate the effect of evolving based on novelty versus fitness, they are fixed at these values in the experiments in this chapter. However, modulatory neurons still affect the learning rate at Hebbian synapses as usual. For a more detailed description of the implications of different coefficient values for the generalized Hebbian plasticity rule, see Niv et al. (2002).

3.3 The Discrete T-Maze Domain

The first domain is based on experiments performed by Soltoggio et al. (2008) on the evolution of neuromodulated networks for the T-Maze learning problem. T-Mazes are often studied in the context of operant conditioning of animals; they are also studied to assess the ability of plastic ANNs (Blynel and Floreano, 2003; Soltoggio et al., 2008). This domain is ideal to test the hypothesis that novelty search escapes deception in adaptive domains because it is well-established from prior work (Blynel and Floreano, 2003; Dürr et al., 2008; Soltoggio et al., 2008, 2007) and can be adjusted to be more or less deceptive, as is done in the experiments presented here. Furthermore, it represents a typical reward-based dynamic scenario (i.e. the agent’s actions that maximize reward intake can change during its lifetime), where optimal performance can only be obtained by an adaptive agent. Thus the results

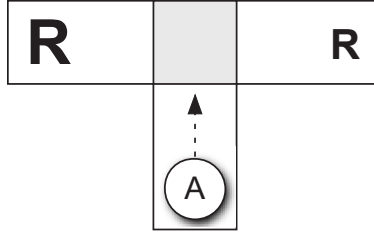


Figure 3.1: **The Discrete T-Maze.** In this depiction, high reward is located on the left and low reward is on the right side, but these positions can change over a set of trials. The goal of the agent is to navigate to the position of the high reward and back home to its starting position. The challenge is that the agent must remember the location of the high reward from one trial to the next.

presented here should also provide more insight into the potential deceptiveness in similar learning problems.

The single T-Maze (Figure 3.1) consists of two arms that either contain a high or low reward. The agent begins at the bottom of the maze and its goal is to navigate to the reward position and return home. This procedure is repeated many times during the agent’s lifetime. One such attempted trip to a reward location and back is called a *trial*. A *deployment* consists of a set of trials (e.g. 20 trials in the single T-Maze experiments in this chapter are attempted over the course of a deployment). The goal of the agent is to maximize the amount of reward collected over deployments, which requires it to memorize the position of the high reward in each deployment. When the position of the reward sometimes changes, the agent should alter its strategy accordingly to explore the other arm of the maze in the next trial. In Soltoggio’s original experiments (Soltoggio et al., 2008), the reward location changes at least once during each deployment of the agent, which fosters the emergence of learning behavior.

However, the *deceptiveness* of this domain with respect to the evolution of learning can be increased if the reward location is not changed in all deployments in which the agent is evaluated. For example, an individual that performs well in the 99 out of 100 deployments wherein learning is not required and only fails in the one deployment that requires learning will most likely score a high fitness value. Thus such a search space is highly deceptive to evolving learning and the stepping stones that ultimately lead to an adaptive agent will not be rewarded. The problem is that learning domains often have the property that significant improvement in fitness is possible by discovering hidden heuristics that avoid lifetime adaptation entirely, creating a pathological deception against learning to learn.

If adaptation is thus only required in a small subset of deployments, the advantage of an adaptive individual over a non-adaptive individual (i.e. always navigating to the same side) in fitness is only marginal. The hypothesis is that novelty search should outperform fitness-based search with increased domain deception.

It is important to note that the agent acts in a discrete grid-like world in the T-Maze experiments presented in this chapter and also in Chapter 4; the agent moves at a constant pace with only a limited number of available discrete actions (e.g. turn left, turn right). Chapter 6 will investigate a more realistic scenario in which the agent has to operate in a continuous T-Maze domain that requires it to develop obstacle avoidance together with the ability to adapt its behavior to changing reward locations. This later advance will be enabled by adaptive ES-HyperNEAT's ability to evolve more complex plastic ANNs.

3.4 T-Maze Experiment

To compare the performance of NEAT with fitness-based search and NEAT with novelty search, each agent is evaluated on ten deployments, each consisting of 20 trials. The number of deployments in which the high reward is moved after ten trials varies among one (called the *1/10 scenario*), five (called the *5/10 scenario*), and ten (called the *10/10 scenario*), effectively controlling the level of deception. The high reward always begins on the left side at the start of each deployment.

Note that all deployments are deterministic, that is, a deployment in which the reward does not switch sides will always lead to the same outcome with the same ANN. Thus the number of deployments in which the reward switches is effectively a means to control the proportional influence of adaptive versus non-adaptive deployments on fitness and novelty. The question is whether the consequent deception impacts novelty as it does fitness.

Of course, it is important to note that a population rewarded for performance in the 1/10 scenario would not necessarily be expected to be attracted to a general solution. At the same time, a process like novelty search that continues to find new behaviors should ultimately encounter the most general such behavior. Thus the hypothesized advantage of novelty search in such scenarios follows naturally from the dynamics of these different types of search.

Figure 3.2 shows the inputs and outputs of the ANN (following Soltoggio et al. (2008)). The *Turn* input is set to 1.0 when a turning point is encountered. *M-E* is set to 1.0 at the end of

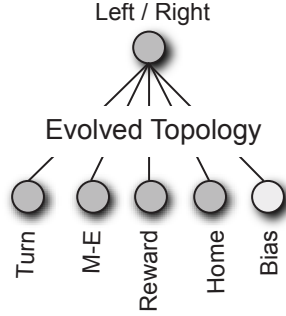


Figure 3.2: **T-Maze ANN Topology.** The network has four inputs, one bias, and one output neuron. *Turn* is set to 1.0 at a turning point location. *M-E* is set to 1.0 at the end of the maze, and *Home* is set to 1.0 when the agent returns to the home location. The *Reward* input returns the level of reward collected at the end of the maze. The bias neuron emits a constant 1.0 activation that can connect to other neurons in the ANN. Network topology is evolved by NEAT.

the maze and *Home* becomes 1.0 when the agent successfully navigates back to its starting position. The *Reward* input is set to the amount of reward collected at the maze end. An agent crashes if it does not (1) maintain a forward direction (i.e. activation of output neuron between -0.3 and 0.3) in corridors, (2) turn either right ($o > 0.3$) or left ($o < -0.3$) when it encounters the junction, or (3) make it back home after collecting the reward. If the agent crashes then the current trial is terminated.

The fitness function for fitness-based NEAT (which is identical to Soltoggio et al. 2008) is calculated as follows: Collecting the high reward has a value of 1.0 and the low reward is worth 0.2. If the agent fails to return home by taking a wrong turn after collecting a reward then a penalty of 0.3 is subtracted from fitness. On the other hand, 0.4 is subtracted if the agent does not maintain forward motion in corridors or does not turn left or right at a

junction. The total fitness of an individual is determined by summing the fitness values for each of the 20 trials over all ten deployments.

Novelty search on the other hand requires a *novelty metric* to distinguish between different behaviors. The novelty metric for this domain distinguishes between learning and non-learning individuals and is explained in more detail in the next section.

3.4.1 Measuring Novelty in the Single T-Maze

The aim of the novelty metric is to measure differences in behavior. In effect, it determines the behavior-space through which the search explores. Because the goal of this chapter is to evolve adaptive individuals, the novelty metric must distinguish a learning agent from a non-learning agent. Thus it is necessary to characterize behavior so that different such behaviors can be compared. The behavior of an agent in the T-Maze domain is characterized by a series of trial outcomes (i.e. 200 trial outcomes for ten deployments with 20 trials each). To observe learning behavior, and to distinguish it from non-learning behavior, it is necessary to run multiple trials in a single lifetime, such that the agent’s behavior before and after a reward switch can be observed. Importantly, the behavior space in the T-Maze domain is therefore significantly larger than in prior experiments (Lehman and Stanley, 2008), effectively testing novelty search’s ability to succeed in a high-dimensional behavior space of 200 dimensions (versus only two dimensions in Lehman and Stanley (2008)). It is important to note that

Trial Outcome			Pairwise Distances	
Name	Collected Reward	Crashed		
NY	none	yes		
LY	low	yes	} 1	} 2
HY	high	yes		
LN	low	no	} 1	} 2
HN	high	no		} 3

Figure 3.3: **The T-Maze Novelty Metric.** Each trial is characterized by (1) the amount of collected reward (2) whether the agent crashed. The pairwise distances (shown at right) among the five possible trial outcomes, *NY*, *LY*, *HY*, *LN*, and *HN*, depend on their behavioral similarities.

the dimensionality of the behavior space is not the only possible characterization of the dimensionality of the problem. For example, the dimensionality of the solution ANN is also significantly related to the difficulty of the problem.

Each trial outcome is characterized by two values: (1) the amount of reward collected (*high*, *low*, *none*) and (2) whether or not the agent crashed. These outcomes are assigned different *distances* to each other depending on how similar they are (Figure 3.3). In particular, an agent that collects the high reward and returns home successfully without crashing (*HN*) should be more similar to an agent that collects the low reward and also returns home (*LN*) than to one that crashes without reaching any reward location (*NY*). The novelty distance metric $dist_{novelty}$ is ultimately computed by summing the distances between each trial outcome of two individuals over all deployments.

Figure 3.4 depicts outcomes over several trials of three example agents. The first agent always alternates between the left and the right T-Maze arm, which leads to oscillating

	Reward Switch								Fitness
Agent 1	LN	HN	LN	HN	HN	LN	HN	LN	4.8
$\text{dist}_n(a_1, a_2) = 1 + 0 + 1 + 0 + 1 + 0 + 1 + 0 = 4.0$									
Agent 2	HN	HN	HN	HN	LN	LN	LN	LN	4.8
Agent 3	HN	HN	HN	HN	LN	HN	HN	HN	7.2

Time \rightarrow

Figure 3.4: **Three T-Maze Sample Behaviors.** These learning and non-learning individuals all exhibit distinguishable behaviors when compared over multiple trials. Agent three achieves the desired adaptive behavior. The vertical line indicates the point in time that the position of the high reward changed. While agents 1 and 2 look the same to fitness, novelty search notices their difference, as the distance calculation (inset line between agents 1 and 2) shows.

low and high rewards. The second agent always navigates to the left T-Maze arm. This strategy results in collecting the high reward in the first four trials and then collecting the low reward after the reward switch. The third agent exhibits the desired learning behavior and is able to collect the high reward in seven out of eight trials. (One trial of exploration is needed after the reward switch.) Interestingly, because both agents one and two collect the same amount of high and low reward, they achieve the same fitness, making them indistinguishable to fitness-based search. However, novelty search discriminates between them because $\text{dist}_{\text{novelty}}(\text{agent}_1, \text{agent}_2) = 4.0$ (Figure 3.4). Recall that this behavioral distance is part of the novelty metric (Equation 2.9), which replaces the fitness function and estimates the sparseness at a specific point in behavior space.

Importantly, fitness and novelty both use the same information (i.e. the amount of reward collected and whether or not the agent crashed) to explore the search space, though in a completely different way. Thus the comparison is fair.

3.4.2 Generalization Performance

An important goal of the comparison between fitness and novelty is to determine which learns to adapt most efficiently in different deployment scenarios, e.g. 1/10, 5/10, and 10/10. Thus it is important to note that, because performance on different scenarios will vary based on the number of trials in which the reward location switches, for the purpose of analyzing the results there is a need for an independent measure that reveals the overall adaptive capabilities of each individual.

Therefore, to test the ability of the individuals to generalize independently of the number of deployments in which the position of the high reward changes, they are tested for 20 trials on each of two different initial settings: (1) high reward starting left and (2) high reward starting right. In both cases, the position of the high reward changes after 10 trials. An individual passes the *generalization test* if it can collect the high reward and return back home in at least 18 out of 20 trials from both initial positions. Two low reward trials in each setting are necessary to explore the T-Maze at the beginning of each deployment and when the position of the high reward switches.

The generalization measure does not necessarily correlate to fitness. An individual that receives a high fitness in the 1/10 scenario can potentially perform poorly on the generalization test because it does not exhibit adaptive behavior. Nevertheless, generalization performance does follow a general upward trend over evaluations and reveals the ultimate quality of solu-

tions (i.e. individuals passing the generalization test would receive high fitness scores in all scenarios).

The number of nearest neighbors for the novelty search algorithm is 15 (following Lehman and Stanley 2008, 2011b). The novelty threshold is 2.0. This threshold for adding behaviors to the archive dynamically changes every 1,500 evaluations. If no new individuals are added during that time the threshold is lowered by 5%. It is raised by 20% if the number of individuals added is equal to or higher than four. The novelty scores of the current population are reevaluated every 100 evaluations to keep them up to date (the archive does not need to be reevaluated). Connection weights range within $[-10, 10]$. These parameter values are shared by all experiments in this chapter. Additional experimental parameters for this experiment and all other experiments in this dissertation are given in the Appendix.

3.5 Single T-Maze Results

Because the aim of the experiment is to determine how quickly a general solution is found by fitness-based search and novelty search, an agent that can solve the generalization test described in Section 3.4.2 counts as a solution.

Figure 3.5 shows the average performance (over 20 runs) of the current best-performing individuals on the generalization test across evaluations for novelty search and fitness-based

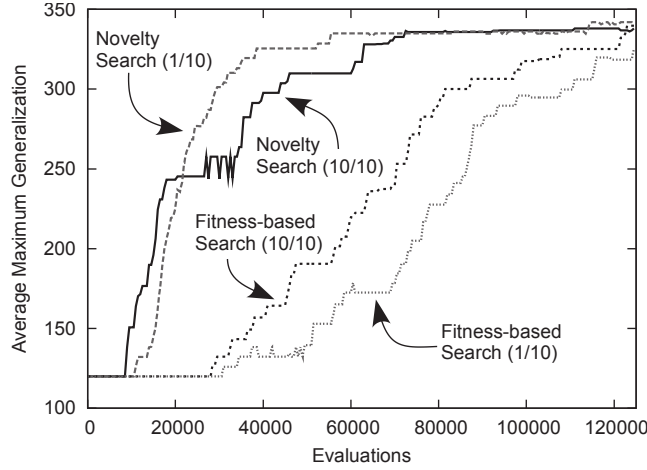


Figure 3.5: **Comparing Generalization of Novelty Search and Fitness-based Search.** The change in performance (calculated like fitness) over evaluations on the generalization test is shown for NEAT with novelty search and fitness-based search in the 1/10 and 10/10 scenarios. All results are averaged over 20 runs. The main result is that novelty search learns a general solution significantly faster.

search, depending on the number of deployments in which the reward location changes. Novelty search performs consistently better in all scenarios. Even in the 10/10 domain that resembles the original experiment (Soltoggio et al., 2008), it takes fitness significantly longer to reach a solution than novelty search. The fitness-based approach initially stalls, followed by gradual improvement, whereas on average novelty search rises sharply from early in the run.

Figure 3.6 shows the average number of evaluations (over 20 runs) that it took fitness-based and novelty-based NEAT to solve the generalization test in the 1/10, 5/10, and 10/10 scenarios. If no solution was found within the initial 125,000 evaluations, the current simulation was restarted (i.e. a new run was initiated). This procedure was repeated until a solution was found, counting all evaluations over all restarts.

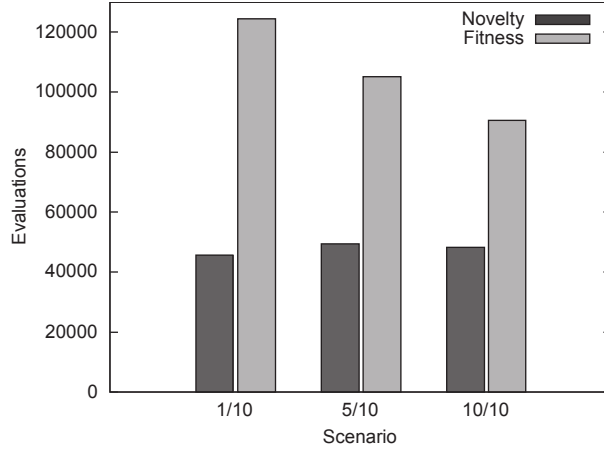


Figure 3.6: **Average Evaluations to Solution for Novelty Search and Fitness-based Search.** The average number of evaluations over 20 runs that it took novelty search and fitness-based search to solve the generalization test is shown. Novelty search performs significantly better in all scenarios and fitness-based search performs even worse when deception is high. Interestingly, novelty search performance does not degrade at all with increasing deception.

Both novelty and fitness-based NEAT were restarted three times out of 20 runs in the 10/10 scenario. Fitness-based search took on average 90,575 evaluations ($\sigma = 52,760$) while novelty search was almost twice as fast at 48,235 evaluations on average ($\sigma = 55,638$). This difference is significant ($p < 0.05$). In the more deceptive 1/10 scenario, fitness-based search had to be restarted six times and it took 124,495 evaluations on average ($\sigma = 81,789$) to find a solution. Novelty search only had to be restarted three times and was 2.7 times faster ($p < 0.001$) at 45,631 evaluations on average ($\sigma = 46,687$).

Fitness-based NEAT performs worse with increased domain deception and is 1.4 times slower in the 1/10 scenario than in the 10/10 scenario. It took fitness on average 105,218 evaluations ($\sigma = 65,711$) in the intermediate 5/10 scenario, which is in-between its performance on

the 1/10 and 10/10 scenarios, confirming that deception increases as the number of trials requiring adaptation decreases. In contrast, novelty search is not significantly affected by increased domain deception: The performance differences among the 1/10, 5/10, and 10/10 scenarios is insignificant for novelty search, confirming its immunity to deception.

Recall that individuals cannot avoid spending at least two trials (when the position of the reward switches for both initial settings; Section 3.4.2) collecting the low reward to pass the generalization test. However, the action *after* collecting the low reward (e.g. crashing into a wall or taking a wrong turn on the way home) is not part of the criteria for passing the test. To ensure that stricter criteria do not change the results, a second experiment consisting of 50 independent runs was performed in which the agent also had to return back home after collecting the *low reward*. That way, the agent always must return home. It took novelty search on average 191,771 evaluations ($\sigma = 162,601$) to solve this harder variation while fitness-based search took 267,830 evaluations ($\sigma = 198,455$). In this harder scenario, both methods needed more evaluations to find a solution but the performance difference is still significant ($p < 0.05$).

Interestingly, novelty search not only outperforms fitness-based search in the highly deceptive 1/10 scenario but also in the intermediate 5/10 scenario and even in the 10/10 scenario, in which the location of the reward changes every deployment. There is no obvious deception in the 10/10 scenario (which resembles Soltoggio’s original experiment; Soltoggio et al. (2008)). However, the recurring plateaus in fitness common to all scenarios (Figure 3.5) suggest a

general problem for evolving learning behavior inherent to dynamic, reward-based scenarios. The next section addresses this issue in more depth.

3.6 Analysis of Deception in the Single T-Maze

Why does novelty search outperform fitness-based search even when the position of the high reward changes in every deployment? To answer this question the analysis in this section is based on runs in a seemingly non-deceptive setting: Every individual is evaluated in one deployment consisting of 20 trials in which the position of the high reward switches after ten trials. The high reward is always located on the left side of the maze at the beginning of each deployment and an individual counts as a solution if it can collect at least 18 out of 20 high rewards and navigate back home. This reduction in deployments combined with the relaxed solution criteria simplifies an in-depth analysis (e.g. by yielding smaller novelty archives that are easy to visualize) and is analogous to the 10/10 scenario. The aim is to uncover the hidden source of deception and how exactly novelty search avoids it.

One interesting way to analyze different evolutionary search techniques is to examine what they consider *best*, i.e. which individuals have the highest chances to reproduce. For novelty search, these are the individuals that display the most novel behavior and therefore enter the archive. For fitness-based search, the most rewarded are the *champions*, i.e. the behaviors with the highest fitness found so far. Although the probabilistic nature of the evolutionary

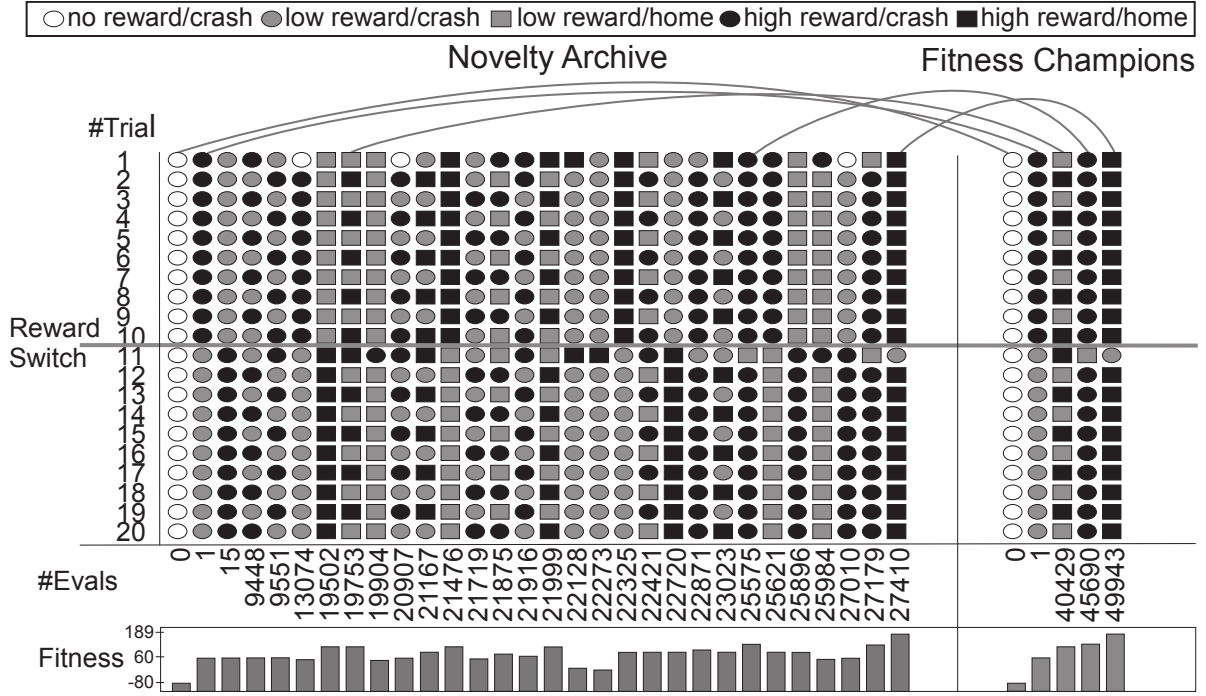


Figure 3.7: **Novelty Search Archive and Fitness Champions.** Behaviors archived by novelty search and the highest-fitness-so-far found by fitness-based search during evolution are shown together with their corresponding fitness and evaluation at which they were discovered. Agents are evaluated on 20 trials and the reward location switches after 10 trials. Arcs (at top) connect behaviors that were highly rewarded by both methods. Novelty search consistently archives new behaviors while fitness-based search improves maximum fitness only four times during the whole evolutionary run. Many of the behaviors found by novelty search would receive the same fitness, which means they are indistinguishable to fitness-based search. The main result is that a higher number of promising directions are explored by novelty search.

search means that such individuals are not guaranteed to produce offspring, they represent the most likely to reproduce.

Highlighting the dramatic difference between these contrasting reward systems, Figure 3.7 shows the behaviors archived by novelty search and the most fit individuals (when they first appear) found by fitness-based search during a typical evolutionary run. It took novelty

search 27,410 evaluations to find a solution in this scenario while fitness-based search took almost twice as long with 49,943 evaluations. While novelty search finds 30 behaviors that are novel enough to enter the archive, fitness only discovers five new champions during the whole evolutionary run. A look at the fitness values of the archived novel behaviors reveals that many of them collapse to the same score, making them indistinguishable to fitness-based search (also see Section 3.4.1 for discussion of such conflation). For example, the second through fifth archived behaviors in Figure 3.7, which represent different combinations of ten *HY* (high reward/crash) and ten *LY* (low reward/crash) events, all receive the same fitness. However, they are all highly rewarded by novelty search at the time they are discovered, which places them into the archive.

In the first 40,429 evaluations, fitness-based search does not discover *any* new champions, giving it little information about the direction in which the search should proceed. On the other hand, novelty search constantly produces novel behaviors *and* takes these behaviors and the current population into account to guide the search.

A visualization technique can help to gain a deeper understanding of how the two approaches navigate the high-dimensional genotypic search space. The most common technique to visualize evolution is to plot fitness over evaluations; although this technique reveals information about the quality of the solution found so far, it provides no information on how the search proceeds through the high-dimensional search space. Various methods have been proposed to illuminate the trajectory of the search (Barlow et al., 2002; Kim and Moon, 2003; Vas-

silev et al., 2000), most of which focus on visualizing the fitness landscape to gain a deeper understanding of its ruggedness.

However, the aim of this analysis is to visualize how the genotypes produced by both search methods traverse the search space *in relation to each other*. Two potential such visualization techniques are *Principal Component Analysis* (PCA) (Kittler and Young, 1973) and *Sammon’s Mapping* (Sammon, 1969). Both methods provide a mapping of high-dimensional points in genotypic space (\mathbb{R}^p) to points in \mathbb{R}^2 . However, while PCA tries to account for the most variance in the data at expense to their original Euclidean distances, Sammon’s Mapping aims to *preserve the distances* of the genotypes in the mapping to a lower dimension (Dybowski et al., 1996). Therefore, Sammon’s mapping is chosen for this analysis because the distances between genotypes produced by fitness-based search and novelty search in the two dimensional visualization should be as close to their original distances as possible to understand how they relate. This approach facilitates the comparison between different regions of the search space that both methods explore.

Sammon’s mapping maps a high-dimensional dataset onto a lower number of dimensions (typically two or three-dimensions), allowing a better understanding of the underlying structure of data. The mapping minimizes the stress measure E , which is the discrepancy between the high dimensional distances δ_{ij} between all objects i and j and the resulting distances d_{ij}

between the data points in the lower dimension:

$$E = \frac{1}{\sum_{i=1}^{n-1} \sum_{j=i+1}^n \delta_{ij}} \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}}. \quad (3.2)$$

The stress measure can be minimized by a steepest descent procedure in which the resulting value of E is a good indicator of the quality of the projection.

For this study, Sammon’s mapping projects high-dimensional *genotypes* produced over the course of evolution onto a two-dimensional space. The output of the mapping are x and y coordinates for every genotype that minimize stress measure E . The original high-dimensional distance δ_{ij} between two genotypes is based on NEAT’s genome distance (Equation 2.4), which is a good indicator of the similarity of two network encodings. The distance d_{ij} between two objects i and j in the visualization space is calculated by their Euclidean distance $\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}$. To make the two-dimensional visualization clearer, not all genotypes created during evolution are part of the Sammon’s mapping; instead, only those are shown that have either (1) a genome distance δ greater than 9.0 compared to already recorded genotypes or (2) have a distance smaller than 9.0 but display a different behavior (based on the novelty metric described in Section 3.4.1). These criteria ensure that a representative selection of genotypes is shown that is still sparse enough to be visible in the projection onto two dimensions.

Figure 3.8 shows a Sammon’s mapping of 882 genotypes; 417 were found by novelty search and 465 were found by fitness-based search during a typical evolutionary run of each. In this

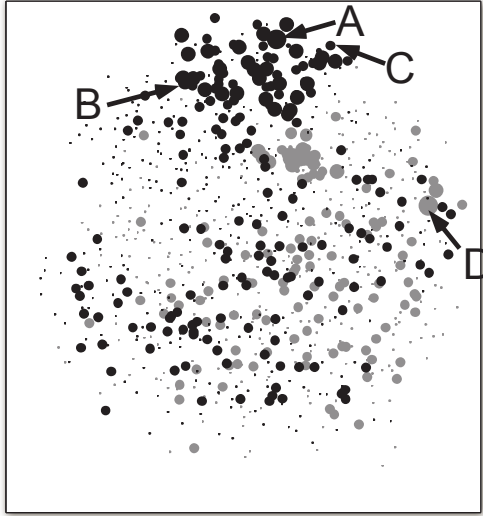


Figure 3.8: **Combined Sammon's Mapping.** The Sammon's mapping of 417 genotypes found by novelty search (gray) and 465 found by fitness-based search (black) is shown. The size of each mark corresponds to the fitness of the decoded network. Larger size means higher fitness. Fitness-based search covered more of the *genotypic* search space than novelty search because it searches through many identical behaviors (though different genotypes) when it is stuck at a local optimum. Four important individuals are identified: the final solution found by fitness-based search (*A*), a network that collects the high reward in the first ten trials and then the low reward (*B*), a network that collects the low reward in 18/20 trials (*C*) and the final solution found by novelty search (*D*). Although *A* and *C* are close they have significantly different fitnesses. Thus while the discovery of *C* could potentially serve as a stepping stone for novelty search, fitness-based search is led astray from the final solution. Points *B* and *D* are discussed in the text.

example, novelty search found a solution after 19,524 evaluations while it took fitness-based search 36,124 evaluations. The low stress measure $E = 0.058$ indicates that the original genotypic distances have been conserved by the mapping. Genotypes that are close to each other in the two-dimensional output space are also close to each other in genotype space.

The mapping reveals that both methods discover different regions of high fitness and that the majority of behaviors simply crash without collecting any rewards (denoted by the smallest points). The main result is that while novelty search (light gray) discovers a genotypic region of high fitness and then quickly reaches the solution (i.e. a behavior that can collect the high reward in at least 18 out of 20 trials, denoted by D in Figure 3.8), fitness-based search (black) needs to cover more of the *genotypic* search space because it searches through many identical behaviors (though different genotypes) when it is stuck at a local optimum.

Interestingly, an intermediate solution found by fitness-based search discovers a behavior that collects 18 out of 20 *low* rewards and returns back home (denoted by C in Figure 3.8). The network that produces this behavior and the final solution (A) are close in genotypic space though they have very different fitness values (178 vs. 50). Thus while the discovery of this behavior could potentially serve as a stepping stone to finding the final solution for novelty search, rather than helping fitness it actually *deceives* it. Agents that collect the high reward in 10 out of 20 trials and return back home (B in Figure 3.8) receive a higher fitness than C -type agents even though they are actually *farther* away from the final solution in genotype space and therefore might lead fitness search astray.

Figure 3.9 examines the temporal progression of the two search methods in more detail by showing the Sammon’s mapping from Figure 3.8 at different stages of evolution in the corresponding run. For each evaluation (i.e. snapshot in time) the mapping shows the genotypes

found so far together with the behaviors archived by novelty search and the champions found by fitness-based search.

Novelty search explores a wider sampling of the search space than fitness-based search during the first 1,000 evaluations. After that, both methods explore similar behaviors until novelty search finds a novel behavior at evaluation 13,912 that collects the low reward and returns back home in the first ten trials and then collects the high reward and returns back home in the successive trials. The ability to successfully return back home after collecting a reward turns out to be a stepping stone to regions of higher fitness. It opens up a wide range of possible new behaviors that lead novelty search to discover 18 new archive members between evaluations 15,000 and 19,520. Interestingly, all the underlying network encodings for these behaviors are close to each other in genotypic space even though they produce significantly different behaviors. Finally, novelty search discovers a solution after 19,524 evaluations.

In contrast, fitness-based search is not able to exploit the same set of behaviors as potential stepping stones because many collapse to the same fitness. While fitness-based search discovers two new champions in the first 1,000 evaluations, it does not discover the next until evaluation 19,520. This more fit behavior is located within a cluster of high fitness genotypes close to the final solution. However, it takes fitness-based search another 17,439 evaluations to discover that solution. The problem again is that fitness-based search is deceived by genotypes that have a higher fitness than those that are actually closer to the solution (Figure 3.8).

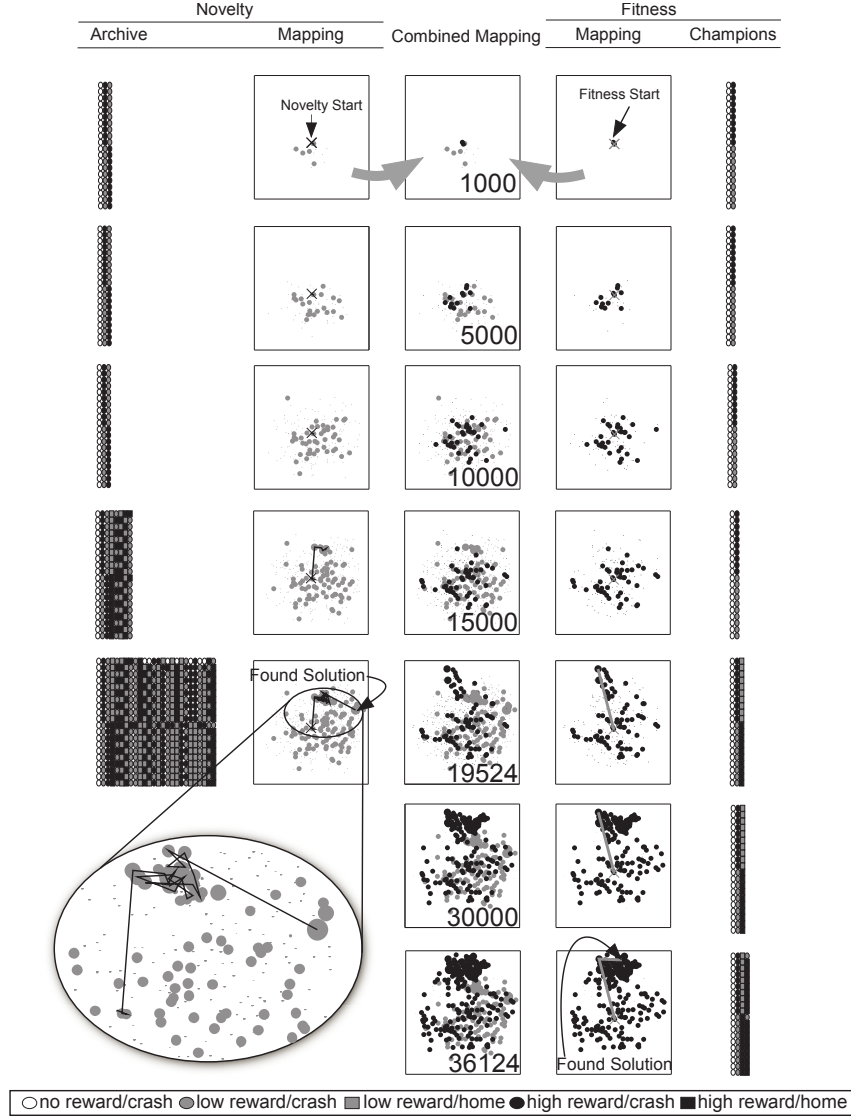


Figure 3.9: **Sammon’s Mapping of Novelty and Fitness-based Search at Different Stages of Evolution.** A mapping of 882 recorded genotypes – 470 produced by novelty search (second column) and 465 by fitness-based search (fourth column) – is shown at seven different time steps together with the corresponding behavior characterizations added to the archive by novelty search and those of the champions found by fitness-based search. Larger markers in the Sammon’s mapping denote higher fitness received by the decoded network. The archived behaviors found by novelty search and the champions found by fitness-based search are connected to show the progression of each search. The magnification (bottom left) of the novelty mapping shows a region of the genotypic space with many novel behaviors that have small genotypic distances to each other. Novelty search finds a solution significantly faster than fitness-based search by exploiting intermediate stepping stones to guide the search.

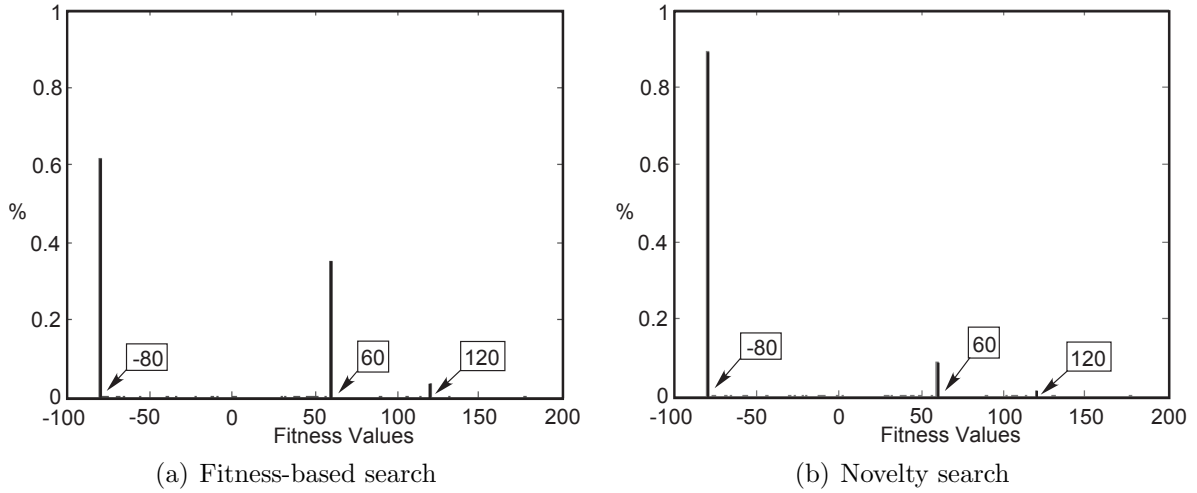


Figure 3.10: Distribution of Fitness for Novelty and Fitness-based Search for a Typical Run. The fitness values for fitness-based search (a) and novelty search (b) both have three peaks in both histograms. There are other values, but they are relatively very rare. Thus many of the genotypes collapse to the same few fitness values, suggesting a lack of intermediate stepping stones for a fitness-based search method.

In a sense, novelty search proceeds more systematically, discovering a region of novel behaviors and then discovering the final solution in fewer evaluations than fitness-based search by exploiting intermediate stepping stones to guide the search. In fact, the number of archived behaviors is always higher than the number of new champions found by fitness across all runs.

To gain a better understanding of the fitness landscape in this domain, Figure 3.10 shows histograms of fitness values for individuals discovered by novelty and fitness-based search in a typical run. The histograms are normalized so that the area sum is one. Interestingly, the vast majority of behaviors (for novelty and fitness-based search) receive one of three different fitness values resulting in three peaks in each distribution. In effect, many behaviors receive

the same fitness, which is another indicator of the lack of intermediate stepping stones and the absence of a fitness gradient in the T-Maze domain. Moreover, the majority of behaviors (61% for fitness and 88% for novelty) simply crash without collecting any reward, suggesting that the encoded networks are brittle to small mutations.

Overall, the analysis in this section shows that novelty search is able to return more information about how behavior changes throughout the search space. It finds a solution significantly faster than fitness-based search by exploiting intermediate stepping stones to guide its search. Interestingly, genotypes that are potential stepping stones for novelty search can lead fitness-based search astray if fitness does not correlate with distance to the final solution (Figure 3.8).

3.7 Additional Experiments

To further demonstrate novelty search’s ability to efficiently evolve plastic ANNs, two substantially more complex scenarios are investigated, which are explained in the next sections.

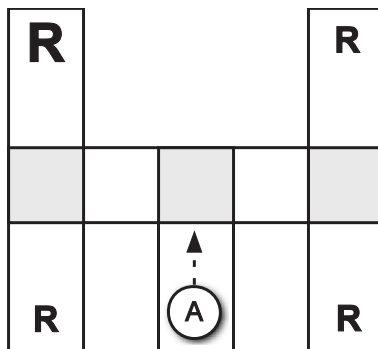


Figure 3.11: **The Double T-Maze.** The maze includes two turning points and four maze endings. In this depiction, high reward is located at the top-left of the maze, but its position can change over a set of trials. The goal of the agent is to navigate to the position of the high reward and remember that location from one trial to the next.

3.7.1 Discrete Double T-Maze

The double T-Maze (Figure 3.11) includes two turning points and four maze endings, which makes the learning task substantially more difficult than the single T-Maze studied in the previous sections (Soltoggio et al., 2008). In effect the agent must now memorize a location on a map that is twice as large.

The experiment follows the setup described in Section 3.4 with a slightly modified novelty metric to capture behaviors in the larger environment. The behavior of an agent is still characterized by a series of trial outcomes, but each such outcome is now determined by the corresponding trial fitness value (e.g. 0.2 for collecting the low reward). The behavioral difference between two behaviors is then calculated as the sum over all trial differences. Each

evaluation consists of two deployments with 200 trials each in which the high reward changes location after every 50 trials. Thus the behavior characterization includes 400 dimensions.

Fitness-based search had to be restarted five times and found a solution in 801,798 evaluations on average ($\sigma=695,534$). Novelty search found a solution in 364,821 evaluations on average ($\sigma=411,032$) and had to be restarted two times. Therefore, even with an increased behavioral characterization (200-dimensional for the single T-Maze vs. 400-dimensional for the double T-Maze) and increased domain complexity, novelty search still finds the appropriate adaptive behavior significantly faster than fitness-based search ($p < 0.05$).

3.7.2 Foraging Bee

Another domain studied for its reinforcement learning-like characteristics is the bee foraging task (Soltoggio et al., 2007; Niv et al., 2002). A simulated bee needs to remember which type of flower yields the most nectar in a stochastic environment wherein the amount of nectar associated with each type of flower can change.

The bee flies in a simulated three-dimensional environment (Figure 3.12a) that contains a 60×60 meter flower field on the ground with two different kind of flowers (e.g. yellow and blue). The bee constantly flies downward at a speed of 0.5m/s and can perceive the flower

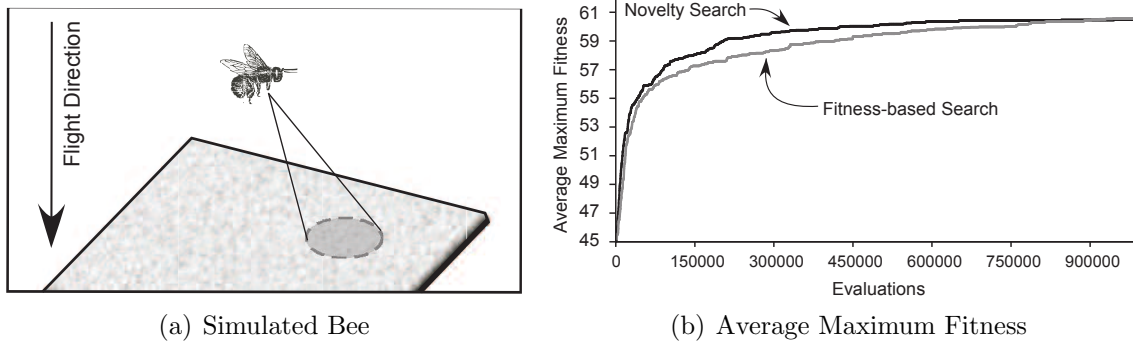


Figure 3.12: **Comparing Novelty Search to Fitness-based Search in the Bee Domain.** The simulated bee flying in a three-dimensional space is shown in (a). The bee is constantly flying downwards but can randomly change its direction. The bee can perceive the flower patch with a simulated view cone. (b) The change in fitness over time (i.e. number of evaluations) is shown for NEAT with novelty search and fitness-based NEAT, which are both averaged over 25 runs for each approach. The main result is that both methods reach about the same average fitness but novelty search finds a solution significantly faster.

patch through a single eye with a 10-degree view cone. The outside of the flower patch is perceived as gray-colored.

The ANN controller has five inputs (following Soltoggio et al. 2007); the first three are *Gray*, *Blue*, and *Yellow*, which receive the percentage of each perceived color in the bee’s view cone. The *Reward* input is set to the amount of nectar consumed after the bee successfully lands on either a yellow or blue flower. It remains zero during the flight. The *Landing* input is set to 1.0 upon landing. The bee has only one output that determines if it stays on its current course or changes its direction to a new random heading.

Each bee is evaluated on four deployments with 20 trials each. The flower colors are inverted after 10 trials on average and between deployments, thereby changing the association between

color and reward value. The amount of collected nectar (0.8 for high-rewarding and 0.3 for low-rewarding flowers) over all trials is the performance measure for fitness-based search.

To mitigate noisy evaluation in novelty search (i.e. a single random change in direction at the beginning of the flight can lead to a very different sequence of collected high and low rewards), each deployment is described by two values. The first is the number of high rewards collected before the color switch and the second is the number of high rewards collected afterwards. This novelty metric rewards the bee for collecting a different number of high rewards before and after the reward switch than other individuals. This measure reflects that what is important is displaying adaptive behavior that is dependent on the time of the reward switch. The coefficients of the generalized Hebbian learning rule for this experiment are $A = -0.79$, $B = 0.0$, $C = 0.0$, and $D = -0.038$. These values worked well for a neuromodulated ANN in the foraging bee domain described by Soltoggio et al. (2007). The other experimental parameters are kept unchanged.

A bee counts as a solution if it displays the desired learning behavior of associating the right color with the currently high-rewarding flower (which corresponds to a fitness of 61). Both fitness-based search and novelty search discovered solutions in 13 out of 25 runs. Novelty search took on average 261,098 evaluations ($\sigma=130,926$) when successful and fitness-based search on average 491,221 evaluations ($\sigma=277,497$). Although novelty search still finds a solution significantly faster ($p < 0.05$), both methods quickly reach a high local optimum before that (Figure 3.12b).

3.8 Implications of Combining Novelty Search with Plasticity

Novelty search outperforms fitness-based search in all domains investigated and is not affected by increased domain deception. This result is interesting because it is surprising that without any other a priori knowledge an algorithm that is not even aware of the desired behavior would find such behavior at all, let alone in a general sense.

Fitness-based search also takes significantly more evaluations to produce individuals that exhibit the desired adaptive behavior when the impact of learning on the fitness of the agent is only marginal. Because it is easier at first to improve fitness without evolving the ability to learn, objective-based search methods are likely to exploit domain-dependent static (i.e. non-adaptive) heuristics.

In the T-Maze domain in this chapter, agents initially learn to always navigate to one arm of the maze and back, resulting in collecting 20 high rewards (i.e. ten high rewards for each of the two starting positions) on the generalization test. Yet, because the reward location changes after ten trials for both initial settings, to be more successful requires the agents to exhibit learning behavior.

The natural question then is why novelty search outperforms fitness-based search in the seemingly non-deceptive 10/10 scenario? While the deception in this setting is not as obvious, the analysis presented in Section 3.6 addressed this issue in more depth. The problem is that evolving the right neuromodulated dynamics to be able to achieve learning behavior is not

an easy task. There is little information available to incentivize fitness-based search to pass beyond static behavior, making it act more like random search. In other words, the stepping stones that lead to learning behavior are hidden from the objective approach behind long plateaus in the search space.

This analysis reveals that fitness-based search is easily led astray if fitness does not reward the stepping stones to the final solution, which is the case in the T-Maze learning problem (Figure 3.8). Novelty search, on the other hand, escapes the deceptive trap and instead builds on the intermediate stepping stones to proceed through the search space more efficiently. Novelty search's ability to keep track of already-explored regions in the search space is probably another factor that accounts for its superior performance.

While in some domains the fitness gradient can be improved, i.e. by giving the objective-based search clues in which direction to search, such an approach might not be possible in dynamic, reward-based scenarios. The problem in such domains is that reaching a certain fitness level is relatively easy, but any further improvement requires sophisticated adaptive behavior to evolve from only sparse feedback from an objective-based performance measure. That is, novelty search returns more *information* about how behavior changes throughout the search space.

In this way, novelty search removes the need to carefully design a domain that fosters the emergence of learning because novelty search on its own is capable of doing exactly that. The only prerequisite is that the novelty metric is constructed such that learning and non-learning

agents are separable, which is not necessarily easy, but is worth the effort if objective-based search would otherwise fail.

In fact, because NEAT itself employs the *fitness sharing* diversity maintenance technique (Goldberg and Richardson, 1987; Stanley and Miikkulainen, 2002) within its species (Section 2.3), the significant difference in performance between NEAT with novelty search and NEAT with fitness-based search also suggests that traditional diversity maintenance techniques do not evade deception as effectively as novelty search. Interestingly, novelty search has also been shown to succeed independently of NEAT (Mouret, 2009) in evolving ANNs and it also outperforms fitness-based search in genetic programming (Lehman and Stanley, 2010a). Thus evidence is building for its generality.

Novelty search’s ability to build gradients that lead to stepping stones is evident in performance curves (Figure 3.5). The increase in generalization performance is steeper than for fitness-based NEAT, indicating a more efficient climb to higher complexity behaviors. In effect, by abandoning the objective, the stepping stones come into greater focus (Lehman and Stanley, 2008, 2011b). Although it means that the search is wider, the alternative is to be trapped by deception.

Of course, there are likely domains for which the representation is not suited to discovering the needed adaptive behavior or in which the space of behaviors is too vast for novelty search to reliably discover the right one. However, even in the double T-Maze domain in which the length of the behavioral characterization is substantially larger (i.e. 400 dimensions), novelty

search still significantly outperforms fitness-based search. There are only so many ways to behave and therefore the search for behavioral novelty becomes computationally feasible and is different than random search. On the other hand, even though novelty search is still significantly faster in the foraging bee task, fitness-based search reaches a local optimum that is very close to the final solution in about the same number of evaluations. A possible explanation for the more even performance in this domain is that the noisy environment offers a vast space of exploitable behavioral strategies.

Overall, the results in this chapter are important because research on evolving adaptive agents has been hampered largely as a result of the deceptiveness of adaptive tasks. Yet the promise of evolving plastic ANNs is among the most intriguing in artificial intelligence. After all, our own brains are the result of such an evolutionary process. Therefore, a method to make such domains more amenable to evolution has the potential to further unleash a promising research direction that is only just beginning. In Chapter 7 this technique helps to evolve solutions in a challenging legged locomotion domain.

Characterizing when and for what reason novelty search fails is also an important future research direction. Yet its performance in experiments in past research (Lehman and Stanley, 2008, 2010a,b; Mouret, 2009) and also in the evolution of legged locomotion in this dissertation (Chapter 7) has proven surprisingly robust. While it is not always going to work well, this chapter suggests that it is a viable new tool in the toolbox of evolutionary computation to counteract the deception inherent in evolving adaptive behavior.

However, evolving neural plasticity is not just limited by deception. A significant problem with most approaches that are based on direct encodings (like NEAT) is that the parameters governing the plasticity for each synapse must be discovered separately by evolution. The next chapter introduces a method called adaptive HyperNEAT that addresses this problem, and shows that plasticity in artificial brains can be distributed in a geometric pattern, which reflects the intuition that synaptic plasticity in biological brains is not encoded in DNA separately for every synapse in the brain.

CHAPTER 4

APPROACH PART I: ADAPTIVE HYPERNEAT

The main idea in adaptive HyperNEAT is that CPPNs in HyperNEAT can not only encode connectivity patterns but also *patterns of plasticity rules*. As in the brain, different *regions* of the ANN should be more or less plastic and employ different learning rules, which HyperNEAT allows because it sees the ANN geometry. The experiments in this chapter are published in Risi and Stanley (2010).

4.1 Indirectly Encoding Local Learning Rules

In general, a learning rule changes the weight of a connection based on presynaptic activity o_i , postsynaptic activity o_j , and the current connection weight w_{ij} :

$$\Delta w_{ij} = \Phi(o_i, o_j, w_{ij}) . \quad (4.1)$$

In the experiments in this chapter three different adaptive HyperNEAT models are introduced and compared that are able to encode different levels of learning rule generality. The goal of this comparison is to elucidate the advantages and disadvantages of different levels

of generality to modeling dynamic learning processes. All three models allow learning rules to be distributed as patterns across the connectivity of an ANN.

The most general **iterated model** (figure 4.1a) augments the four-dimensional CPPN that normally encodes connectivity patterns with three additional inputs: presynaptic activity o_i , postsynaptic activity o_j , and the current connection weight w_{ij} . That way, the synaptic plasticity of a connection between two two-dimensional points (x_1, y_1) and (x_2, y_2) can be described by

$$\Delta w_{ij} = \text{CPPN}(x_1, y_1, x_2, y_2, o_i, o_j, w_{ij}) . \quad (4.2)$$

The update of the synaptic weights can thereby be iteratively performed by the same CPPN that normally encodes network connectivity, which allows evolving increasingly complex learning rules. In effect, the CPPN encodes an entire dynamical system, including how changes depend on both location *and* activity. The CPPN is *queried* on every tick of the clock to update the ANN weights. The initial weight configuration is determined by querying the CPPN as in the original HyperNEAT approach (Section 2.4.2) with the presynaptic activity, postsynaptic activity, and weight inputs all set to zero.

The less general **Hebbian ABC model** augments the CPPN instead with four additional *outputs* (figure 4.1b): learning rate η , correlation term A , presynaptic term B , and postsynaptic term C . When the CPPN is initially queried, these parameters are permanently stored, which allows the synaptic weight to be modified during the lifetime of the agent by

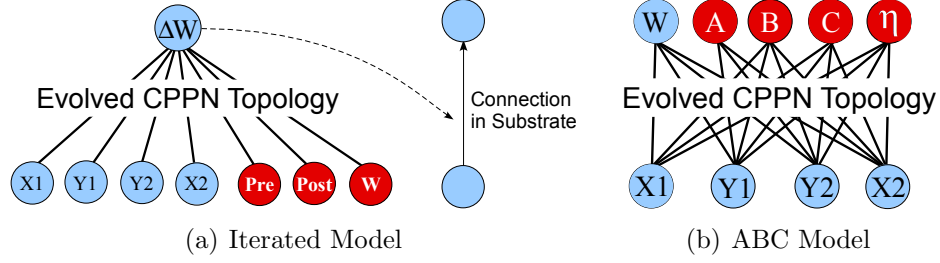


Figure 4.1: **Adaptive HyperNEAT**. CPPNs for the iterated (a) and ABC models (b) are shown. The CPPN in (a) is continually requeried during the lifetime of the agent to determine the weight change given the location of the connection, activation of the presynaptic and postsynaptic neuron, and the current weight as input. In contrast, the CPPN in (b) is only activated once to determine the three parameters A – C and the learning rate η , which control synaptic plasticity during the lifetime of the agent, in addition to the initial weight w .

the following plasticity rule:

$$\Delta w_{ij} = \eta \cdot [A o_i o_j + B o_i + C o_j] . \quad (4.3)$$

Traditional approaches to evolving adaptive ANNs with direct encodings, like the one in the previous chapter, also normally evolve the coefficients of equation (4.3) but because of the limitations of direct encodings often only employ *one* such evolved rule throughout all ANN connections (Soltoggio, 2008; Soltoggio et al., 2008). The difference here is that A , B , C , and η are indirectly encoded by HyperNEAT in a geometric pattern across the connectivity of the whole network. Therefore each connection can potentially employ a different rule if necessary. However, unlike the more general iterated model, this CPPN only produces variants of the ABC Hebbian rule. Thus the space of possible rules is more restricted.

Finally, the simplest model is **plain Hebbian**. The CPPN has only one additional output that encodes the learning rate η :

$$\Delta w_{ij} = \eta \cdot o_i o_j . \quad (4.4)$$

This variant tests for the minimal sufficient dynamics to solve the T-Maze domain given in this dissertation, which is explained in the next section. For all adaptive HyperNEAT models synaptic strength is bound within the range $[-1.0, 1.0]$. Experimental parameters are given in the Appendix.

4.2 Discrete T-Maze Experiment

The discrete T-Maze presented here (figure 4.2a) is similar to the setup described in Section 3.3. However, because HyperNEAT can see the geometry of the underlying task, the robot has slightly different input and output sensors (figure 4.2b) and their locations are designed to geometrically correlate (e.g. seeing something on the left correlates to turning left). Thus the CPPN can exploit the geometry of the agent. Also, instead of a *Reward* input, the agent has a *Color* input (explained shortly), which is set to the color of the collected reward at the maze end and determines the amount of reward given to the agent.

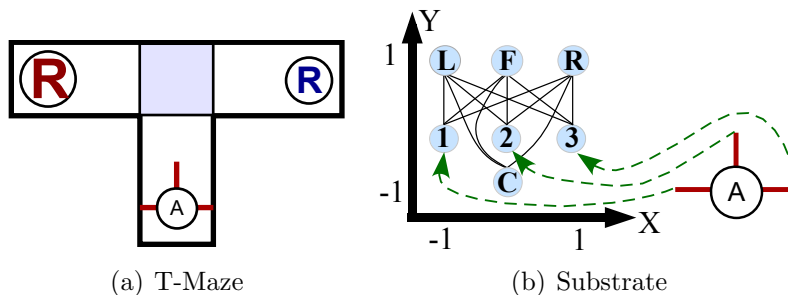


Figure 4.2: T-Maze and Substrate Configuration. (a) In this depiction, high reward is located on the left and low reward is on the right side, but these positions can change over a set of trials. The challenge for the agent is to remember the location of the high reward from one trial to the next. (b) The autonomous agent A is equipped with three distance sensors and a reward color sensor that is set to zero during navigation.

The agent is equipped with three rangefinder sensors that detect walls to the left, front, and right of the robot. The three output neurons are *Left*, *Forward*, and *Right*. At each simulated time step, the agent continues to move straight at a constant pace if the *Forward* output has the highest activation level. Otherwise the agent turns 90 degrees in the direction of the highest activated neuron (Left or Right). An agent crashes if it does not (1) maintain a forward direction in corridors or (2) turn either right or left when it encounters the junction. If the agent crashes then the current trial is terminated.

Two T-Maze scenarios are studied to elucidate the advantages and disadvantages of encoding different levels of plasticity rule generality. **Scenario 1** resembles the traditional T-Maze domain described in Section 3.3. Each agent is evaluated on four deployments with 100 trials each. The starting position of the high reward alternates between deployments and switches



Figure 4.3: **Nonlinear Reward Color Encoding.** The agent receives a high reward for green or red and a low reward for blue or yellow. The ANN color encoding together with the given ANN topology requires the agent to include a nonlinear learning rule.

positions after 50 trials on average. Color input values of 1.0 and 0.1 encode the high (red) and low (blue) reward, respectively.

In **scenario 2**, the agent is exposed to a total of four different colored rewards. The first deployment resembles scenario 1 with reward signatures of 0.1 and 1.0. However, in the *second* deployment, color input values of 0.3 and 0.8 are introduced to encode new high yellow and low green rewards, respectively (figure 4.3). Adding these intermediate reward colors yields a reward signature that is *not linearly separable*. Because the ANN controlling the agent does not have any hidden neurons, the learning rule must *itself* be nonlinear. Scenario 2 therefore makes a good domain for this study because it requires evolving a specific learning rule that depends on the ANN topology.

The fitness function, which is the same for all compared approaches and identical to Soltoggio et al. (2008), is calculated as follows: Collecting a high reward has a value of 1.0 and a low reward is worth 0.2. A penalty of 0.4 is subtracted if the agent does not maintain forward motion in corridors or does not turn left or right at a junction. The total fitness of an individual is determined by summing the fitness values for each of the 100 trials over all deployments.

Note that although Chapter 3 showed that novelty search significantly outperforms fitness-based search in the traditional T-Maze domain, a standard fitness function is employed in this chapter to keep the experiment focused on the issue of adaptation.

4.3 Discrete T-Maze Results

The standard T-Maze (scenario 1) is solved when the agent reaches a fitness of 395. A minimum amount of exploration (i.e. collecting the low reward) is required at the beginning of each deployment and when the reward positions switch. The T-Maze with nonlinear reward signature (scenario 2), consisting of two deployments with different reward signatures, is solved with a fitness of 195. All reported results are averaged over 20 runs.

Figure 4.4a shows the average training performance over generations for the standard T-Maze (scenario 1). It took the ABC model 141 generations ($\sigma=141$) on average to find a solution. The iterated model took 89 generations ($\sigma=61$) on average. While the fitness for the iterated model initially increases more slowly than for ABC, it finds a solution slightly (though not significantly) faster on average. The plain Hebbian model cannot solve the task. Although both the more general iterated model and the ABC model can solve the task, the iterated model is computationally more expensive because the CPPN must be continually requeried for every ANN connection.

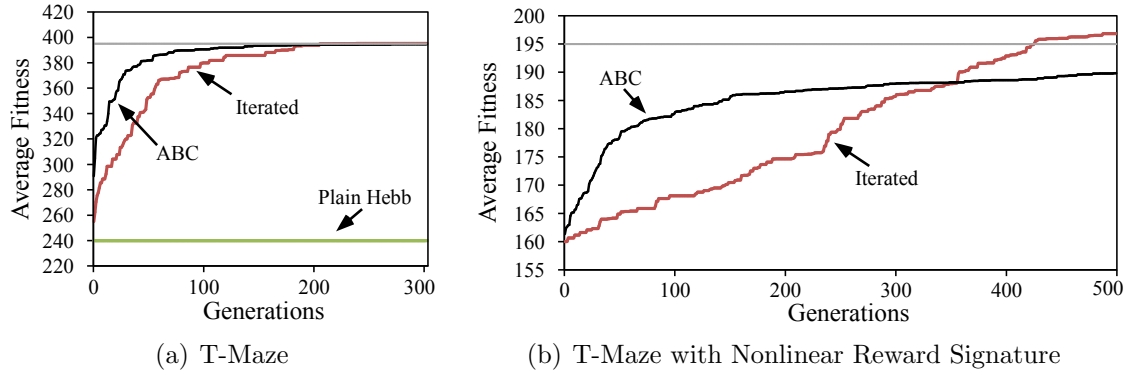


Figure 4.4: T-Maze Training Performance. The change in performance over evaluations for both scenarios is shown in this figure. All results are averaged over 20 runs. The horizontal line (top) indicates at what fitness the domain is solved. The iterated and ABC model are both able to solve the standard T-Maze domain (a) in about the same number of generations whereas the plain Hebbian approach does not show the necessary dynamics. The T-Maze domain with a nonlinear reward signature (b) requires a nonlinear learning rule, which only the iterated model discovers.

The average training performance over generations for scenario 2 is shown in figure 4.4b. The plain Hebbian rule is not tested in this variant because it is not able to solve the standard T-Maze. Whereas the iterated model solves the domain in 19 out of 20 runs, in 367 generations ($\sigma=101$) on average, ABC is not able to solve the task with the given ANN topology, which suggest the need for a nonlinear learning rule in this scenario (or potentially an ANN with hidden nodes). The more general iterated model is able to evolve such a rule.

Figure 4.5a shows CPPN-encoded learning rules of an ANN solution discovered by the iterated model. The function embodied by the CPPN (figure 4.5b) encodes a geometric pattern of nonlinear learning rules. Interestingly, the evolved rules resemble postsynaptic-based learning rules that have been shown essential in the T-Maze domain (Soltoggio, 2008).

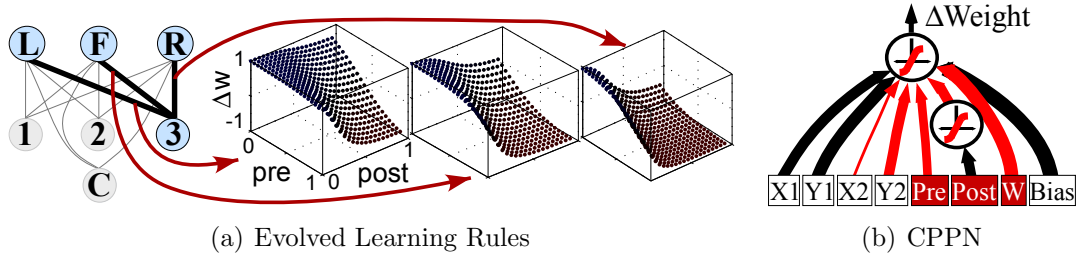


Figure 4.5: **Discovered Learning Rules of an ANN Solution Created by the Iterated Model and the Underlying CPPN.** The nonlinear learning rules shown in (a) are encoded by the evolved CPPN shown in (b). A geometric pattern of learning rules can be seen that varies with the target node’s x location. The amount of synaptic change is a function of the pre- and postsynaptic activity and the corresponding positions of the pre- and postsynaptic neurons in the substrate (weight input w on the CPPN is set to zero in this depiction).

The conclusion is that adaptive HyperNEAT can in fact efficiently indirectly encode local plasticity rules which is a step towards more biologically plausible adaptive systems.

4.4 Adaptive HyperNEAT Implications

The indirect HyperNEAT encoding is able to generate ANNs with millions of connections based on underlying geometric motifs (Stanley et al., 2009). This chapter introduced an extension called adaptive HyperNEAT that generates not only patterns of weights across the connectivity of an ANN, but also patterns of learning rules (figure 4.5).

While the ABC model together with an adequate ANN topology should be sufficient for most domains, the nonlinear variant of the T-Maze learning domain reveals that sometimes

a more general encoding may be necessary. Although the ANN topology could potentially have been extended (or automatically determined as the unified approach introduced in Chapter 6 will attempt) to allow a less general model to solve the nonlinear T-Maze domain, this experiment confirms the risk of making a priori assumptions about the type of necessary learning rules (Yao, 1999).

However, the generality of the indirect encoding of plasticity trades off with its computational cost. The most general iterated model is computationally expensive because the CPPN must be continually requiered for every ANN connection. The computational complexity for every time step is $O(n) + nO(m)$, where $O(n)$ and $O(m)$ are the costs of simulating an ANN with n connections and an underlying CPPN with m connections, respectively. Thus the most general model in its current form might be too computationally expensive for practical purposes that require large CPPNs and ANNs. However, it gives us a reference point from which to derive more specialized models such as the ABC model.

In the current iterated model the synaptic weights are updated at every time step. Characterizing how often a weight update is necessary is an important future research direction that may allow cutting down the computational cost of even the most general model.

While the ANN substrate in this chapter did not rely on any hidden nodes, more challenging domains will likely depend on more complicated network structures. As the next chapter will show, not only can HyperNEAT be extended to represent patterns of learning rules, but also to determine the placement of the hidden nodes solely based on implicit information in

the CPPN-generated pattern. Ultimately these two abilities will be combined in Chapter 6, which will allow the unified approach to evolve increasingly complex neural geometry, density, and plasticity.

CHAPTER 5

APPROACH PART II: EVOLVABLE-SUBSTRATE HYPERNEAT

The placement of nodes in original HyperNEAT is decided by the user. Yet whereas it is often possible to determine how sensors and effectors relate to domain geometry, it is difficult for the user to determine the best placement and number of necessary hidden nodes *a priori*. For example, the location of the hidden nodes in the substrate in figure 2.2 had to be decided by the user. HyperNEAT thus creates the strange situation that it can decide with what weight any two nodes in space should be connected, but it cannot tell us anything about where the nodes should be. What representation can evolve the placement and density of nodes that can potentially span between networks of several dozen nodes and several billion?

This chapter explains how the internal geometry of node placement and density can be determined only based on implicit information in an infinite-resolution pattern of weights in a method called evolvable-substrate HyperNEAT. The text in this chapter is a major extension of work previously published at the Genetic and Evolutionary Computation Conference (GECCO 2011; Risi and Stanley 2011). The initial ES-HyperNEAT approach was published at GECCO 2010 (Risi et al., 2010b). The next section introduces the key insight, which is that a representation that encodes the pattern of connectivity across a network automatically contains implicit information on where the nodes should be placed.

5.1 Implicit Information in the Hypercube

In HyperNEAT the pattern of connectivity is described by the CPPN, where every point in the four-dimensional space denotes a potential connection between two two-dimensional points (recall that a *point* in the four-dimensional hypercube is actually a *connection weight* and not a node). Because the CPPN takes x_1, y_1, x_2 , and y_2 as input, it is a function of the *infinite* continuum of possible coordinates for these points. In other words, the CPPN encodes a potentially infinite number of connection weights within the hypercube of weights. Thus one interesting way to think about the hypercube is as a theoretically infinite pattern of possible connections that *might* be incorporated into a neural network substrate. If a connection is chosen to be included, then by necessity the nodes that it connects must *also* be included in the substrate. Thus by asking which connections to include from the infinite set, we are also asking which nodes to include.

By shifting the question of what to include in the substrate from nodes to connections, two important insights follow: First, the more such connections are included, the more nodes would also be added to the substrate. Thus the *node density* increases with the number of connections. Second, for any given infinite-resolution pattern, there is some sampling density above which increasing density further offers no advantage. For example, if the hypercube is a uniform gradient of maximal connection weights (i.e. all weights are the same constant), then in effect it encodes a substrate that computes the same function at every node. Thus adding more such connections and nodes adds no new *information*. On the other

hand, if there is a stripe of differing weights running through the hypercube, but otherwise uniform maximal connections everywhere else, then that stripe contains information that would contribute to a *different* function from its redundantly-uniform neighbors.

The key insight is thus that it is not always a good idea to add more connections because for any given finite pattern, at some resolution there is no more *information* and adding more weights at such high resolution would be redundant and unnecessary. This maximal useful resolution varies for different regions of the hypercube depending on the complexity of the underlying weight pattern in those regions. Thus the answer to the question of which connections should be included in ES-HyperNEAT is that connections should be included at high enough resolution to capture the detail (i.e. information) in the hypercube. Any more than that would be redundant. Therefore, an algorithm is needed that can choose many points to express in regions of high variance and fewer points to express in regions of relative homogeneity. Each such point is a connection weight in the substrate whose respective nodes will be expressed as well. The main principle is simple: *Density follows information*. In this way, the placement of nodes in the topographic layout of an ANN is ultimately a signification of where information is stored within weights.

To perform the task of choosing points (i.e. weights) to express, a data structure is needed that allows space to be represented at variable levels of granularity. One such multi-resolution technique is the *quadtree* (Finkel and Bentley, 1974), which traditionally describes two-dimensional regions. It has been applied successfully in fields ranging from pattern recogni-

tion to image encoding (Rosenfeld, 1980; Strobach, 1991) and is based on recursively splitting a two-dimensional region into four sub-regions. That way, the decomposition of a region into four new regions can be represented as a subtree whose parent is the original region with one descendant for each decomposed region. The recursive splitting of regions can be repeated until the desired resolution is reached or until no further subdivision is needed because additional resolution is no longer uncovering any new information. The next sections describe the ES-HyperNEAT algorithm in more detail. Pseudocode for the algorithm can be found in the Appendix.

5.2 Quadtree Information Extraction

Instead of searching directly in the four-dimensional hypercube space (recall that it takes four dimensions to represent a two-dimensional connectivity pattern), ES-HyperNEAT iteratively discovers the ANN connections starting from the inputs and outputs of the ANN (which are predefined by the user). This approach focuses the search within the hypercube on two-dimensional cross-sections of the hypercube.

The foundation of the ES-HyperNEAT algorithm is the quadtree information extraction procedure, which receives a two-dimensional position as input and then either analyzes the *outgoing* connectivity pattern from that single neuron (if it is an input), or the *incoming* connectivity pattern (if it is an output). For example, given an input neuron at (a, b) ,

the quadtree connection-choosing algorithm is applied only to the two-dimensional outgoing connectivity patterns described by the function $CPPN(a, b, x2, y2)$, where $x2$ and $y2$ range between -1 to 1. The algorithm works in two main phases (figure 5.1): In the **division and initialization phase** the quadtree is created by recursively subdividing the initial square (lines 8–11 of Algorithm 1 in Appendix), which spans the space from (-1, -1) to (1, 1), until a desired initial resolution r is reached (e.g. 4×4 , which corresponds to a quadtree depth of 3). For every quadtree square with center (x, y) the CPPN is queried with arguments (a, b, x, y) and the resulting connection weight value w is stored (line 14 of Algorithm 1).

Given the values (w_1, w_2, \dots, w_k) of the k leaf nodes in a subtree of quadtree node p and mean weight \bar{w} , the variance of node p in the quadtree can be calculated as $\sigma_p^2 = \frac{1}{k} \sum_1^k (\bar{w} - w_i)^2$. This variance is a heuristic indicator of the heterogeneity (i.e. presence of information) of a region. If the variance of the parent of a quadtree leaf is still higher than a given division threshold d_t (line 20 of Algorithm 1) then the division phase can be reapplied for the corresponding leaf’s square, allowing increasingly high densities. Just as the initialization resolution ensures that some minimum level of sampling is enforced (i.e. the basic shape of the encoded pattern is likely to be discovered), a maximum resolution level r_m can also be set to place an upper bound on the number of possible neurons if desired. However, it is theoretically interesting that in principle this algorithm can yield arbitrarily high density, which means that very large ANNs are possible to represent.

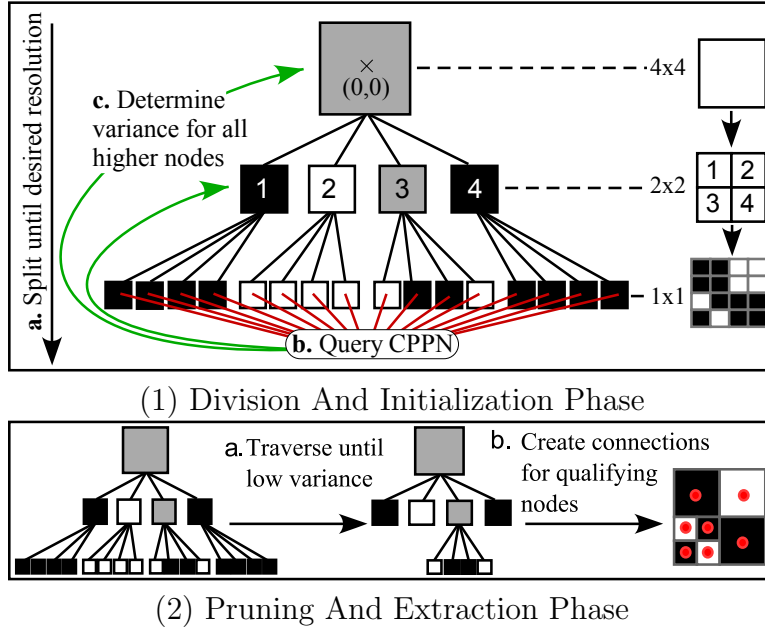


Figure 5.1: **Quadtree Information Extraction Example.** Given an input neuron at (a, b) the algorithm works in two main stages. (1) In the *division and initialization phase* the quadtree is created by recursively splitting each square into four new squares until the desired resolution is reached (1a), while the values (1b) for each square with center (x, y) are determined by $\text{CPPN}(a, b, x, y)$ and the variance values of each higher node are calculated (1c). Gray nodes in the figure have a variance greater than zero. Then, in the *pruning and extraction phase* (2), the quadtree is traversed depth-first until the node's variance is smaller than a given threshold (2a). A connection (a, b, x, y) is created for each qualifying node with center (x, y) (2b). That way, the density of neurons in different regions will correspond to the amount of *information* in that region.

The quadtree representation created in the initialization phase serves as a heuristic variance indicator to decide on the connections (and therefore placement and density of neurons) to express. Because more connections should be expressed in regions of higher variance, a **pruning and extraction phase** (Algorithm 2 in the Appendix) is next executed (figure 5.1 bottom), in which the quadtree is traversed depth-first until the current node’s variance is smaller than the variance threshold σ_t^2 (line 4 of Algorithm 2) or until the node has no children (which means that the variance is zero). Subsequently, a connection (a, b, x, y) is created for each qualifying node with center (x, y) (line 22 of Algorithm 2; the band threshold in Algorithm 2 is explained shortly). The result is higher resolution in areas of more variation.

Figure 5.2a shows an example of the outgoing connections from source neuron $(0, -1)$ (depicted only by their target location for clarity) chosen at this stage of the algorithm. The variance is high at the borders of the circles, which results in a high density of expressed points at those locations. However, for the purpose of identifying connections to include in a neural topography, the raw pattern output by the quadtree algorithm can be improved further. If we think of the pattern output by the CPPN as a kind of *language* for specifying the locations of expressed connections, then it makes sense additionally to prune the points around borders so that it is easy for the CPPN to encode points definitively within one region or another.

Thus a more parsimonious “language” for describing density patterns would ignore the edges and focus on the inner region of *bands*, which are points that are enclosed by at least two

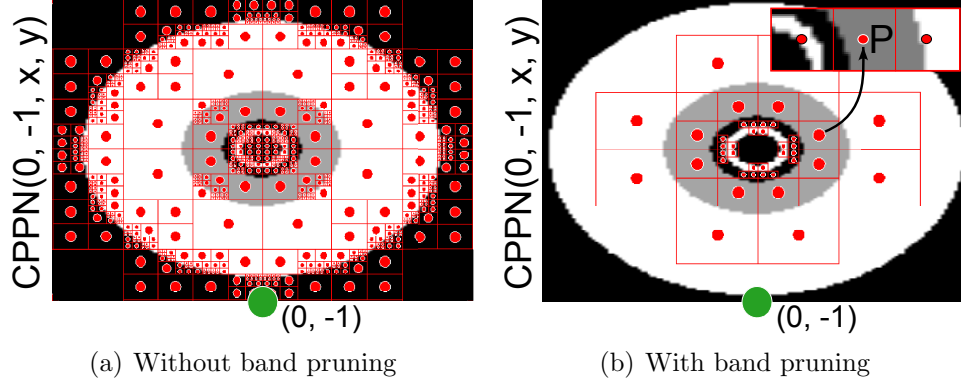


Figure 5.2: **Example Connection Selection.** Chosen connections (depicted only by their target location) originating from $(0, -1)$ are shown in (a) after the pruning stage but without band pruning. Points that still remain after band pruning (e.g. point P , whose neighbors at the same resolution have different CPPN activation levels) are shown in (b). The resulting point distribution reflects the information inherent in the pattern.

neighbors on opposite sides (e.g. left and right) with different CPPN activation levels (figure 5.2b). Furthermore, narrower bands can be interpreted as requests for more point density, giving the CPPN an explicit mechanism for affecting density.

Thus, to facilitate banding, a pruning stage is added that removes points that are not in a band. Membership in a band for a square with center (x, y) and width ω is determined by band level

$$\beta = \max(\min(d_{\text{top}}, d_{\text{bottom}}), \min(d_{\text{left}}, d_{\text{right}})),$$

where d_{left} is the difference in CPPN activation levels between the connection (a, b, x, y) and its left neighbor at $(a, b, x - \omega, y)$ (line 9 of Algorithm 2). The other values, d_{right} , d_{bottom} , and d_{top} , are calculated accordingly. If the band level β is below a given threshold β_t then

the corresponding connections is not expressed (line 19 of Algorithm 2). Figure 5.2b shows the resulting point selections with band pruning.

This approach also naturally enables the CPPN to increase the density of points chosen by creating more bands or making them thinner. Thus no new information and no new representational structure beyond the CPPN already employed in HyperNEAT is needed to encode node placement and connectivity, as concluded in the next section.

5.3 ES-HyperNEAT Algorithm

The complete ES-HyperNEAT algorithm (Algorithm 3 in the Appendix) is depicted in figure 5.3. The connections originating from an input at $(0, -1)$ (figure 5.3a; line 7 in Algorithm 3) are chosen based on the connection-choosing approach described in the previous section. The corresponding hidden nodes are created if not already existent (lines 9–10 in Algorithm 3). The approach can be iteratively applied to the discovered hidden nodes until a user-defined maximum iteration level is reached (line 15 in Algorithm 3) or no more information is discovered in the hypercube (figure 5.3b). To tie the network into the outputs, the approach chooses connections based on each output’s *incoming* connectivity patterns (figure 5.3c; line 27 in Algorithm 3).

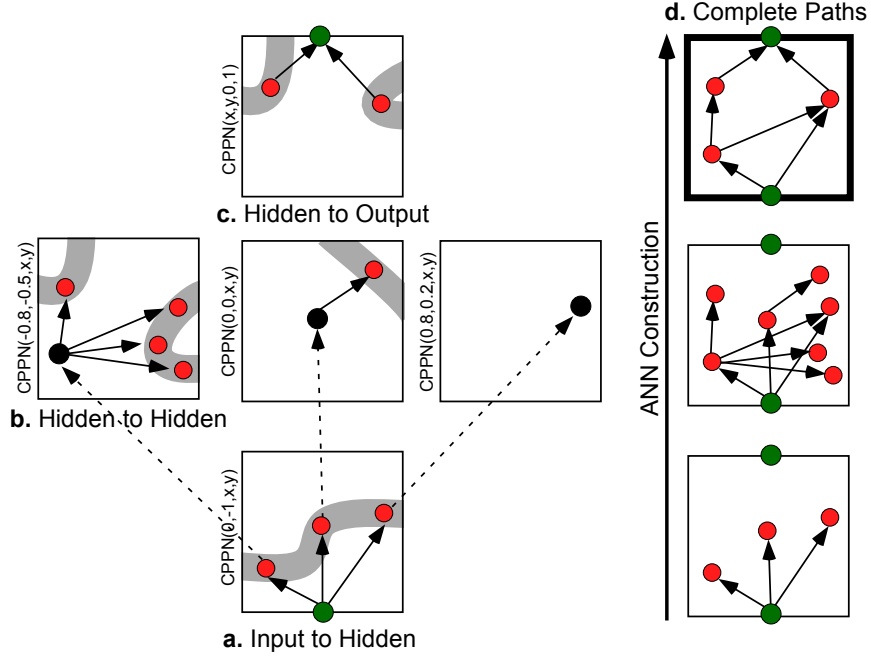


Figure 5.3: **The ES-HyperNEAT Algorithm.** The algorithm starts by iteratively discovering the placement of the hidden neurons from the inputs (a) and then ties the network into the outputs (c). The two-dimensional motif in (a) represent *outgoing* connectivity patterns from a single input node whereas the motif in (c) represent *incoming* connectivity pattern for a single output node. The target nodes discovered (through the quadtree algorithm) are those that reside within bands in the hypercube. In this way regions of high variance are sought only in the two-dimensional cross-section of the hypercube containing the source or target node. The algorithm can be iteratively applied beyond the inputs to the discovered hidden nodes (b). Only those nodes are kept at the end that have a path to an input *and* output neuron (d). That way, the search through the hypercube is restricted to functional ANN topologies.

Once all hidden neurons are discovered, only those are kept that have a path to an input *and* output neuron (figure 5.3d; line 34 in Algorithm 3). This iterated approach helps to reduce computational costs by focusing the search on a sequence of two-dimensional cross-sections of the hypercube instead of searching for information directly in the full four-dimensional hyperspace.

ES-HyperNEAT ultimately unifies a set of algorithmic advances stretching back to NEAT, each abstracted from an important facet of natural evolution that contributes to its ability to evolve complexity. The first is that evolving complexity requires a mechanism to increase the information content in the genome over generations (Stanley and Miikkulainen, 2002, 2004). Second, geometry plays an important role in natural neural connectivity; in neuroevolution, endowing neurons with geometric coordinates means that the genome can in effect project regularities in connectivity across the neural geometry, thereby providing a kind of scaffolding for situating cognitive structures (Stanley et al., 2009; Gauci and Stanley, 2008, 2010). Third, the placement and density of neurons throughout the geometry of the network should reflect the complexity of the underlying functionality of its respective parts (Risi and Stanley, 2011). Later in this dissertation it will be further unified with adaptive HyperNEAT.

Because ES-HyperNEAT can automatically deduce node geometry and density from CPPNs instead of requiring a priori placement (as in original HyperNEAT), it significantly expands the scope of neural structures that evolution can discover. The approach does not only evolve the location of every neuron in the brain, but also can represent regions of varying density,

which means resolution can increase holistically over evolution. The main insight is that the connectivity and hidden node placement can be automatically determined by information already inherent in the pattern encoded by the CPPN. In this way, the density of nodes is automatically determined and effectively unbounded. Thus substrates of unbounded density can be evolved and determined without any additional representation beyond the original CPPN in HyperNEAT.

5.4 Key Hypotheses

Automatically determining the placement and density of hidden neurons introduces several advantages beyond just liberating the user from making such decisions. This section introduces the key hypotheses in this chapter that elucidate these advantages, which the experiments in sections 5.5, 5.6, and 5.7 aim to validate.

Hypothesis 1. ES-HyperNEAT facilitates evolving networks with targeted connectivity for multimodal tasks.

Although it produces regular patterns of weights, the original HyperNEAT tends to produce fully-or near fully-connected networks (Clune et al., 2010), which may create a disadvantage in domains where certain neurons should only receive input from one modality while other neurons should receive inputs from multiple modalities, thus allowing the sharing of

information about the underlying task similarities in the hidden layer. In contrast, because ES-HyperNEAT only creates connections where there is high variance in the hypercube, it should be able to find greater variation in connectivity for different neurons. To test Hypothesis 1, the first experiment will explore how ES-HyperNEAT performs in a multi-task domain (Section 5.5), which requires the agent to react differently based on the type of input (e.g. rangefinder or radar) it receives.

Hypothesis 2. The fixed locations of hidden nodes in original HyperNEAT that are chosen by the user make finding an effective pattern of weights more difficult than by allowing the algorithm itself to determine their locations, as in ES-HyperNEAT.

The problem is that when node locations are fixed, the pattern in the hypercube that is encoded by the CPPN must intersect those node coordinates at precisely the right locations. Even if such a CPPN encodes a pattern of weights that expresses an effective network, a slight shift (i.e. a small translation) of the pattern would cause it to detach from the correct node locations. Thus the network would receive a low fitness even though it actually does encode the right pattern, if only it were slightly shifted. In contrast, ES-HyperNEAT in effect *tracks* shifts in the underlying pattern because the quadtree algorithm searches for the appropriate locations of nodes regardless of exactly where the pattern is expressed. This increased flexibility means that the feasible area of the search space will be larger and hence easier to hit.

Comparing the complexities of the evolved CPPNs will give an indication of how hard it is for the different HyperNEAT variants to represent the correct pattern. Additionally, analyzing the resulting ANNs will demonstrate whether ES-HyperNEAT can express hidden nodes at slightly different locations when the pattern of weights changes.

Hypothesis 3. ES-HyperNEAT is able to elaborate on existing structure by increasing the number of synapses and neurons in the ANN during evolution while regular HyperNEAT takes the entire set of ANN connection weights to represent a partial solution.

The second experiment in a deceptive maze navigation domain (Section 5.6) will isolate this issue by examining the affect of a task with several intermediate milestones on both variants.

Hypothesis 4. ES-HyperNEAT can more easily evolve modular ANNs than the original fixed-substrate HyperNEAT, in part because it facilitates evolving network with limited connectivity and because it has the capability to start the evolutionary search with a bias towards locality and towards certain canonical ANN topologies.

The third experiment, called left & right retina problem (Clune et al., 2010; Kashtan and Alon, 2005) (Section 5.7), will test of the ability to evolve modular structures because it benefits from separating different functional structures.

If these hypothesis are correct then ES-HyperNEAT should not only match but outperform the original HyperNEAT, as the experiments in the next sections will test.

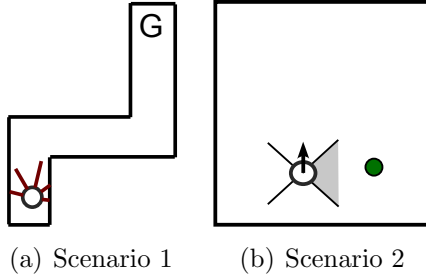


Figure 5.4: **The Dual Task.** In the dual task domain the agent either has to exhibit wall-following (a) or food-gathering behavior (b) depending on the type of sensory input it receives.

5.5 Experiment 1: Dual Task

Organisms in nature have the ability to switch rapidly between different tasks depending on the demands of the environment. For example, a rat should react differently when placed in a maze or in an open environment with a visible food source. The dual task domain presented here (figure 5.4) will test the ability of ES-HyperNEAT and regular HyperNEAT to evolve such task differentiation for a multimodal domain.

The dual task domain consists of two non-dependent scenarios (i.e. the performance in one scenario does not directly influence the performance in the other scenario) that require the agent to exhibit different behaviors and to react either to its rangefinders or pie-slice sensors. Because certain hidden neurons ideally would be responsible for information that should be treated differently, while other hidden neurons should be able to share information where the tasks are similar, this domain will likely benefit from ANNs that are not fully-connected,

which the original HyperNEAT has struggled to produce in the past (Clune et al., 2010). ES-HyperNEAT should facilitate the evolution of networks with more targeted connectivity, as suggested by Hypothesis 1, because connections are only included at a high enough resolution to capture the information in the hypercube.

The first scenario is a simple navigation task in which the agent has to navigate from a starting point to an end point in a fixed amount of time using only its rangefinder sensors to detect walls (figure 5.4a). The fitness in this scenario is calculated as $f_{\text{nav}} = 1 - d_g$, where d_g is the distance of the robot to the goal point at the end of the evaluation scaled into the range $[0, 1]$. The second scenario is a food gathering task in which a single piece of food is placed within a square room with an agent that begins at the center (figure 5.4b). The agent attempts to gather as much food as possible within a time limit using only its pie-slice sensors, which act as a compass towards the food item. Food only appears at one location at a time and is placed at another random location once consumed by the agent. The fitness for the food gathering task is defined by: $f_{\text{food}} = \frac{n + (1 - d_f)}{4}$, where n corresponds to the number of collected food items (maximum four) and d_f is the distance of the robot to the next food item at the end of the evaluation.

The total fitness is calculated as the average performance on both scenarios. The domain is considered *solved* when the agent is able to navigate to the goal point in the first scenario and successfully collects all four food items in the second scenario, which corresponds to a fitness of 1.

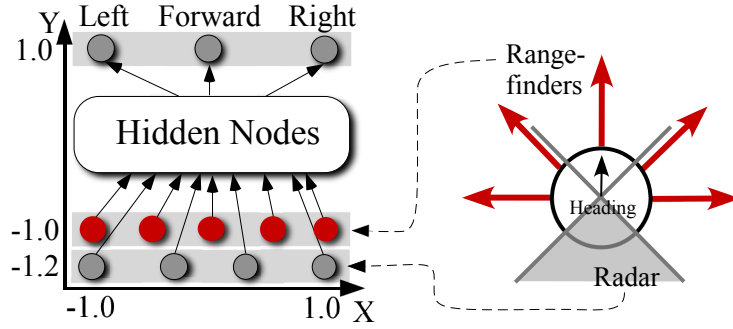


Figure 5.5: **Substrate Configuration and Sensor Layout.** The controller substrate is shown at left. Whereas the number of hidden nodes for the fixed-substrate approach is determined in advance, ES-HyperNEAT decides on the positions and density of hidden nodes on its own. The sensor layout is shown on the right. The autonomous agent is equipped with five distance and four pie-slice sensors.

5.5.1 Experimental Setup

Evolvable and fixed-substrate (original) HyperNEAT use the same placement of input and output nodes on the substrate (figure 5.5), which are designed to correlate senses and outputs geometrically (e.g. seeing something on the left and turning left). Thus the CPPN can exploit the geometry of the agent. The agent is equipped with five rangefinder sensors that detect walls and four pie-slice sensors that act as a compass towards the next food item. All sensor values are scaled into the range $[0,1]$, where lower activation indicates closer proximity to a wall. At each discrete moment of time, the number of units moved by the agent is $20F$, where F is the forward effector output. The agent also turns by $(L - R) * 18^\circ$, where L is the left effector output and R is the right effector output. A negative value is interpreted as a right turn.

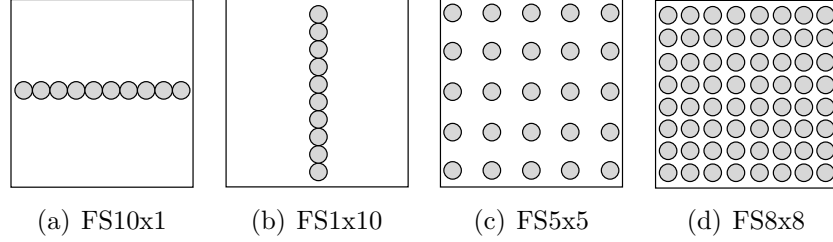


Figure 5.6: **FS-HyperNEAT Hidden Node Layouts.** This figure shows (a) a horizontal configuration of hidden nodes, (b) a vertical arrangement, and two (c,d) grid configurations.

To highlight the challenge of deciding the location and number of available hidden nodes, ES-HyperNEAT is compared to four fixed-substrate variants (figure 5.6). **FS10x1** is the typical setup with a single row of ten hidden neurons in a *horizontal* line at $y = 0$ (figure 5.6a). For the **FS1x10** variant ten hidden neurons are arranged *vertically* at $x = 0$ (figure 5.6b). **FS5x5** has a substrate containing 25 hidden nodes arranged in a 5×5 grid (figure 5.6c). **FS8x8** tests the effects on performance of uniformly increasing the number of hidden nodes from 25 to 64 neurons (figure 5.6d). To generate such a controller for original HyperNEAT, a four-dimensional CPPN with inputs x_1 , y_1 , x_2 , and y_2 queries the substrate shown in figure 5.5, to determine the connection weights between the input/hidden, hidden/output, and hidden/hidden nodes. In contrast, ES-HyperNEAT decides the placement and density of nodes on its own.

Experimental parameters for this experiment and all other experiments in this chapter are given in the Appendix.

5.5.2 Dual Task Results

All results are averaged over 20 runs. Figure 5.7 shows the training performance over generations for the HyperNEAT variants on the dual task domain. ES-HyperNEAT solves the domain in all runs and took on average 33 generations ($\sigma = 31$), whereas the best performing fixed-substrate variant, FS5x5, finds a solution in only 13 out of 20 runs. The difference in average final performance is significant ($p < 0.001$ according to the Student’s t-test).

These results suggest that a multimodal domain benefits from ES-HyperNEAT’s ability to generate networks with limited and targeted connectivity, which an analysis of the evolved ANNs confirms. ES-HyperNEAT usually creates networks in which certain hidden neurons only receive direct input from the agent’s rangefinders, while other neurons receive input from both modalities (e.g. rangefinders and radar). The networks created by the original HyperNEAT are generally fully-or near fully-connected and do not show the same level of targeted connectivity.

Interestingly, the average *CPPN* complexity of solutions discovered by FS5x5 is 9.7 hidden nodes ($\sigma = 6.7$), almost six times higher than CPPN solutions by ES-HyperNEAT, which have 1.65 hidden nodes on average ($\sigma = 2.2$). This difference is significant ($p < 0.05$) and indicates that dictating the location of the hidden nodes a priori makes it harder for the CPPN to represent the correct pattern, as suggested by Hypothesis 2.

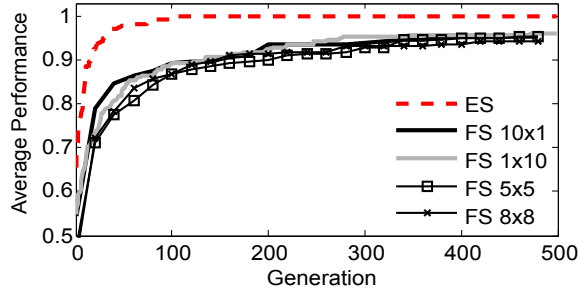


Figure 5.7: **Average Dual Task Performance.** The average best fitness over generations is shown for the dual task domain for the different HyperNEAT variants, which are averaged over 20 runs. The main result is that ES-HyperNEAT significantly outperforms all other variants.

5.6 Experiment 2: Maze Navigation

To evolve controllers for more complicated tasks will require a neuroevolution method that benefits from previously-discovered partial solutions to find the final solution. While direct encodings like NEAT allow the network topology to complexify over generations, leading to increasingly sophisticated behavior, they suffer from the problem of reinvention. That is, even if different parts of the solution are similar they must be encoded and therefore discovered separately.

HyperNEAT alleviated this problem by allowing the solution to be represented as a pattern of parameters, rather than requiring each parameter to be represented individually. However, because regular HyperNEAT tends to produce fully-connected ANNs (Clune et al., 2010), it likely takes the entire set of ANN connection weights to represent a partial task solution, while ES-HyperNEAT should be able to elaborate on existing structure because

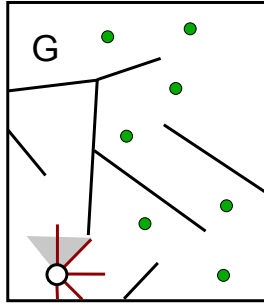


Figure 5.8: **Maze Navigation Domain.** The goal of the agent in the maze navigation domain is to reach goal point G . Because the task is deceptive the agent is rewarded for making incremental process towards the goal by following the waypoints.

it can increase the number of connections and nodes in the substrate during evolution, as suggested by Hypothesis 3.

To test this third hypothesis on when ES-HyperNEAT provides an advantage, a task is needed in which a solution is difficult to evolve without crossing several intermediate milestones. One such task is the deceptive maze navigation domain introduced in Lehman and Stanley (2008). In this domain (figure 5.8), a robot controlled by an ANN must navigate in a maze from a starting point to an end point in a fixed time. The robot has five rangefinders that indicate the distance to the nearest wall within the maze, and four pie-slice radar sensors that fire when the goal is within the pie-slice. The experimental setup follows the one described in Section 5.5.1 with the same substrate configuration and sensor layout (figure 5.5). The agent thus sees walls with its rangefinders and the goal with its radars.

If fitness is rewarded proportionally to how close the robot ends from the goal, cul-de-sacs in the maze that lead close to the goal but do not reach it are deceptive local optima (Lehman

and Stanley, 2011b). Therefore, in this section, the fitness function f rewards the agent explicitly for discovering stepping stones towards the goal:

$$f = \begin{cases} 10, & \text{if the agent is able to reach the goal} \\ n + (1 - d), & \text{otherwise,} \end{cases}$$

where n is the number of passed waypoints (which are *not* visible to the agent) and d is the distance of the robot to the next waypoint scaled into the range $[0, 1]$ at the end of the evaluation. The idea is that agents that can reach intermediate waypoints should make good stepping stones to those that reach further waypoints. ES-HyperNEAT should be able to elaborate more efficiently on agents that reach intermediate waypoints by increasing gradually their neural density.

5.6.1 Maze Navigation Results

ES-HyperNEAT performed significantly better than the other variants in the maze navigation domain ($p < 0.001$) (figure 5.9a) and finds a solution in 19 out of 20 runs in 238 generations on average when successful ($\sigma = 262$). The default setup for FS-HyperNEAT, FS10x1, reaches a significantly higher average maximum fitness than the vertically arrangement FS1x10 ($p < 0.001$) or the grid-like setup FS8x8 ($p < 0.05$).

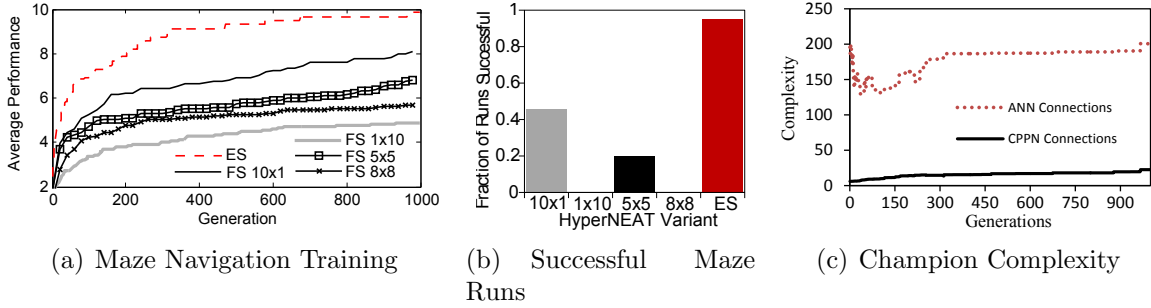


Figure 5.9: **Average Performance and Champion Complexity.** The average best fitness over generations is shown for the maze navigation domain (a) for the different HyperNEAT variants, which are averaged over 20 runs. The fraction of 20 runs that successfully solve the maze navigation domain is shown in (b) for each of the HyperNEAT variants after 1,000 generations. The average number of connections of the champion ANNs produced by ES-HyperNEAT and the number of connections of the underlying CPPNs are shown in (c). Increasing CPPN complexity shows a positive (and significant) correlation with an increase in ANN substrate complexity.

The differing performance of evolvable- and fixed-substrate HyperNEAT can also be appreciated in how frequently they solve the problem perfectly (figure 5.9b). ES-HyperNEAT significantly outperforms all fixed-substrate variants and finds a solution in 95% of the 20 runs. FS10x1 solves the domain in 45% of runs, whereas the vertical arrangement of the same number of nodes (FS1x10) degrades performance significantly ($p < 0.001$), not finding a solution in any of the runs. FS5x5 finds a solution in 20% of all runs. Interestingly, uniformly increasing the number of hidden nodes to 64 for FS8x8, which might be hypothesized to help, in fact degrades performance significantly ($p < 0.001$), not finding a solution in any of the runs.

In ES-HyperNEAT, there is a significant positive correlation ($r = 0.95$, $p < 0.001$ according to the Spearman’s rank correlation coefficient) between the number of connections in the

CPPN and in the resulting ANN (figure 5.9c). This trend indicates that the substrate evolution algorithm may tend to create increasingly complex indirectly-encoded networks even though it is not explicitly designed to do so (e.g. like regular NEAT). The complexity of ANN (substrate) solutions (242 connections on average) is more than 9 times greater than that of the underlying CPPNs (25 connections on average), which suggests that ES-HyperNEAT can encode large ANNs from compact CPPN representations.

The conclusion is that evolving the substrate can significantly increase performance in tasks that require incrementally building on stepping stones. The next two sections examine how this result comes about.

5.6.2 Example Solution Lineage

To gain a better understanding of how an indirect encoding like ES-HyperNEAT elaborates a solution over generations, additional evolutionary runs in the maze navigation domain were performed with sexual reproduction disabled (i.e. every CPPN has only one ancestor). This change facilitates analyzing the lineage of a single champion network. Disabling sexual reproduction did not result in a significant performance difference.

An example of four milestone ANNs in the lineage of a solution and the CPPNs that encode them is shown in figure 5.10. All ANNs share common geometric features: most prominent

are the symmetric network topology and denser regions of hidden neurons resembling the shape of an “H” (except the second ANN). Between generations 24 and 237 the ANN evolves from not being able to reach the first waypoint to solving the task.

The solution discovered at generation 237 shows a clear holistic resemblance to generation 106 despite some general differences. Both networks have strong positive connections to the three output neurons that originate at slightly different hidden node locations. This slight shift is due to a movement of information within the hypercube for which ES-HyperNEAT can nevertheless compensate, as suggested by Hypothesis 2. The number of connections gradually increases from 184 in generation 24 to 356 in generation 237, indicating the incremental elaboration on existing ANN structure, as suggested by Hypothesis 3. Interestingly, the final ANN solves the task without feedback from its pie-slice sensors.

Figure 5.10 also shows that ES-HyperNEAT can encode large ANNs from compact CPPN representations. The solution ANN with 40 hidden neurons and 356 connections is encoded by a much smaller CPPN with only 5 hidden neurons and 18 connections.

In contrast to direct encodings like NEAT (Stanley and Miikkulainen, 2002, 2004), genotypic CPPN mutations can have a more global effect on the expressed ANN patterns. For example, changes in only four CPPN weights are responsible for the change in topology from the second to the third ANN milestone. Other solutions followed similar patterns but single neuron or connection additions to the substrate do also sometimes occur.

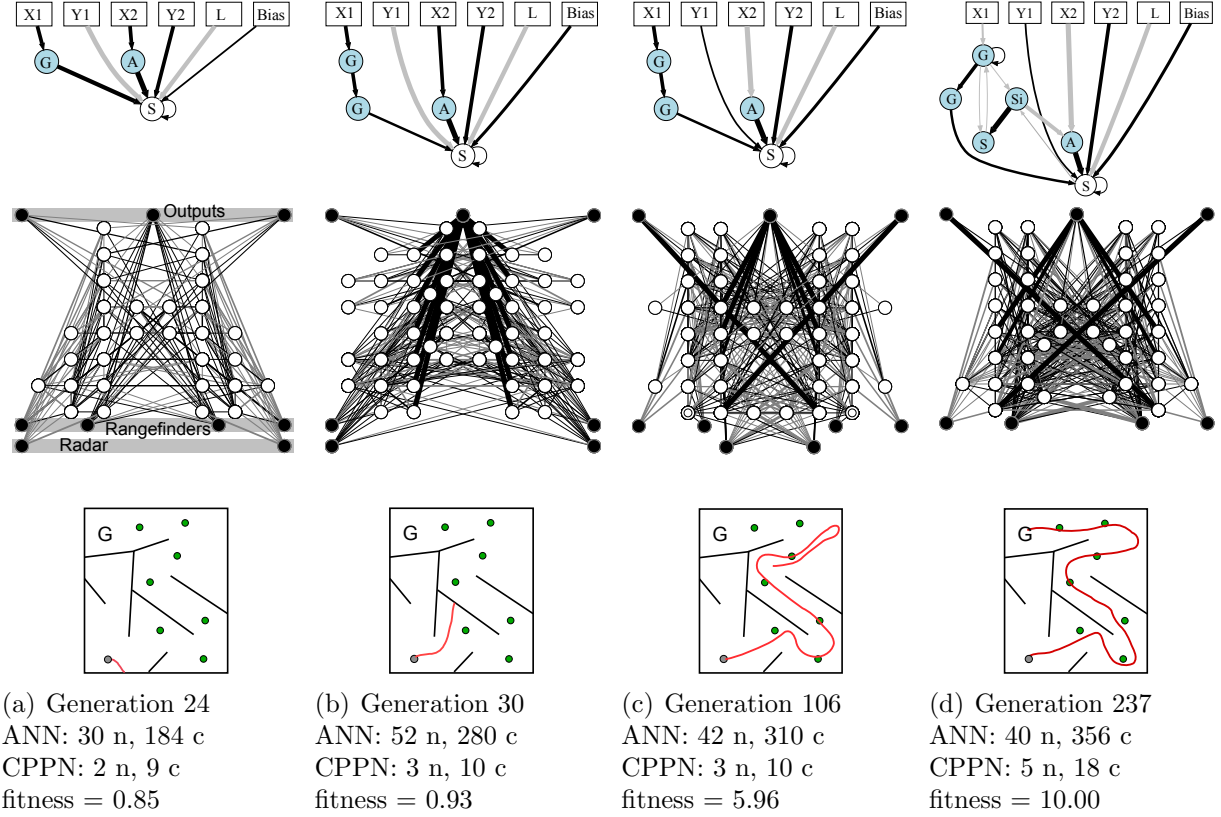


Figure 5.10: ANN Milestones and Underlying CPPNs Together With the Agent's behavior From a Single Maze Solution Lineage. Four ANN milestones (bottom) and the CPPNs (top) that encode them are shown together with the number of hidden neurons n and connections c . Fitness f is also shown. Positive connections are dark whereas negative connections are light. Line width corresponds to connection strength. Hidden nodes with recurrent connections are denoted by a smaller concentric circle. CPPN activation functions are denoted by G for *Gaussian*, S for *sigmoid*, Si for *sine*, and A for *absolute value*. The CPPNs receive the length L of the queried connection as an additional input. The gradual increase of connections indicates an increase of information in the hypercube, which in turn leads to an increase in performance.

5.6.3 Evolvability Analysis

Kirschner and Gerhart (1998) define evolvability as “an organism’s capacity to generate heritable phenotypic variation.” The highly evolvable representations found in biological systems have allowed natural evolution to discover a great variety of diverse organisms. Thus facilitating a representation’s effective search (i.e. its evolvability) is an important research direction in EC (Reisinger et al., 2005).

The dual task (Section 5.5) and the maze navigation domain (Section 5.6) suggest that the original HyperNEAT fails to elaborate on existing ANN structure because it likely consumes the entire set of connection weights to represent an intermediate solution. Furthermore, small mutations in the CPPN can cause a shift in the location of information within the hypercube for which the original HyperNEAT cannot compensate, making such evolved individuals fragile. However, ES-HyperNEAT should be more robust because it can compensate for shifts in the CPPN pattern by following the movement of information within the hypercube (Hypothesis 2). This ability and the fact that ES-HyperNEAT can elaborate on existing ANN structure (Hypothesis 3) should allow ES-HyperNEAT to more easily generate individuals whose offspring display diverse functional behaviors, and thus more heritable phenotypic variation. To demonstrate this capability, the evolvability of the different representations needs to be measured.

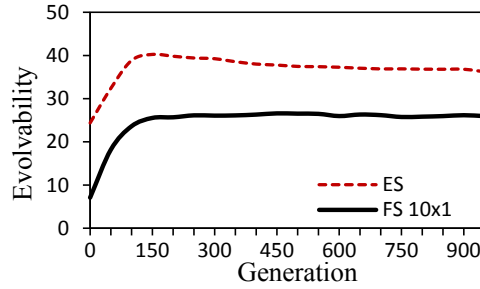


Figure 5.11: **Comparing the Evolvability of Fixed-Substrate and Evolvable-Substrate HyperNEAT.** ES-HyperNEAT exhibits a significantly higher evolvability throughout all generations. Results are averaged over 20 runs.

Kirschner’s definition reflects a growing consensus in biology that the ability to generate phenotypic variation is fundamental to evolvability (Kirschner and Gerhart, 1998; Wagner and Altenberg, 1996; Pigliucci, 2008). Therefore, following Lehman and Stanley (2011a), an individual’s evolvability is estimated by generating many children from it and then measuring the degree of phenotypic variation among those offspring. In effect, this measure quantifies how well the underlying encoding enables behaviorally diverse mutations. To measure phenotypic variation a behavioral distance measure is needed. Following Lehman and Stanley (2011a), behavioral distance in the maze navigation domain is measured in this analysis by the Euclidean distance between the ending position of two individuals. Evolvability is measured every 50 generations, when each individual in the population is forced to create 200 offspring by asexual reproduction. A greedy algorithm calculates the evolvability by adding each individual to a list of behaviors if its behavioral distance to all other individuals already in the list is higher than a given threshold. Thus the number of added behaviors is an indicator of the individual’s ability to generate phenotypic variation (i.e. its evolvability).

Figure 5.11 shows the average evolvability of the best-performing fixed-substrate variant FS10x1 compared to ES-HyperNEAT. The main result is that ES-HyperNEAT shows a significantly ($p < 0.001$) higher evolvability across all generations, further explaining its better performance in the domains presented here.

5.7 Experiment 3: Left & Right Retina Problem

Modularity likely plays an important role in the evolvability of complex biological organisms (Hansen, 2003; Dürr et al., 2010; Clune et al., 2010; Kashtan and Alon, 2005). Lipson (2007) defines functional modularity as the structural localization of function, which allows parts of a solution to be optimized independently (Kashtan and Alon, 2005). Because modularity enhances evolvability and allows natural systems to scale to tasks of high complexity, it is an important ingredient in evolving complex networks, which is a major goal in the field of GDS. ES-HyperNEAT should more easily evolve modular ANNs than the original fixed-substrate HyperNEAT because it has the capability to start the evolutionary search with a bias towards locality and certain canonical ANN topologies, as suggested by Hypothesis 4.

Following Verbancsics and Stanley (2011) and Clune et al. (2010), the modular domain in this section is a modified version of the retina problem, originally introduced by Kashtan and Alon (2005). The goal of the ANN is to identify a set of valid 2×2 patterns on the left and right side of a 4×2 artificial retina (figure 5.12). That is, the ANN must *independently*

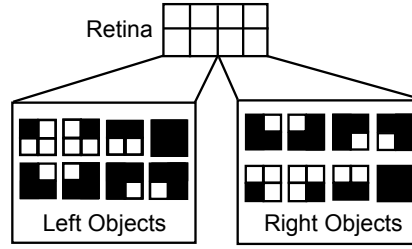


Figure 5.12: **The Retina Problem.** The artificial retina consist of 4×2 pixels that constitute the inputs to the ANN. Eight out of 16 possible patterns are considered left objects. The same is true for the right four pixels, though the eight valid patterns are different.

decide for each pattern presented to the left and right side of the retina if that pattern is a valid left or right pattern, respectively. Thus it is a good test of the ability to evolve modular structures because the left and right problem components are ideally separated into different functional structures.

5.7.1 Link Expression Output

While the original HyperNEAT approach performs poorly in generating modular ANNs (Clune et al., 2010), a recently-introduced extension called Link Expression Output (HyperNEAT-LEO) (Verbancsics and Stanley, 2011) showed that allowing HyperNEAT to evolve the pattern of weights *independently* from the pattern of connection expression, while seeding HyperNEAT with a bias towards local connectivity, allows modular structures to arise naturally.

The LEO is represented as an additional output to the CPPN that indicates whether a connection should be expressed or not. If the LEO output is greater than zero the corresponding connection is created and its weight is set to the original CPPN weight output. Because HyperNEAT evolves such patterns as functions of geometry, important general topographic principles for organizing connectivity can be seeded into the initial population. For example, a key topographic concept in nature that encourages modularity is locality, that is, components of a module are located near each other. Verbancsics and Stanley (2011) showed that by seeding HyperNEAT with a bias towards local connectivity implemented through the LEO, modular structures arise naturally. Furthermore, the authors report that they achieved the best results in the retina left & right task by only seeding with the concept of locality along the x -axis. That is, the seed CPPN starts with a Gaussian node G that receives $x_1 - x_2$ as input and is connected to the LEO output. Therefore G peaks when the two coordinates are the same, seeding the CPPN with a concept of locality. This result makes sense because the retina problem is distributed along the horizontal axis (figure 5.12)

Because ES-HyperNEAT does not require any special changes to the traditional HyperNEAT CPPN, enhancements like the LEO can in principle also be incorporated into ES-HyperNEAT. Thus extending ES-HyperNEAT with a LEO is straightforward and should combine the advantages of both methods: evolving modular ANNs should be possible wherein the placement and density of hidden nodes is determined solely based on implicit information in the hypercube. In this combined approach, once all connections are discovered by the weight-choosing approach (Section 5.3) only those are kept whose LEO output is greater

than zero. In fact, the idea that geometric concepts such as locality can be imparted to the CPPN opens an intriguing opportunity to go further than the LEO locality seed in ES-HyperNEAT. In fact, it is also possible to start the evolutionary search with a bias towards certain ANN topologies, as explained in the next section.

5.7.2 ES-HyperNEAT Geometry Seeding

Because the pattern output by the CPPN in ES-HyperNEAT is a kind of language for specifying the locations of expressed connections and nodes, ES-HyperNEAT can be seeded to *begin* with a bias towards certain ANN topologies (e.g. ANNs with multiple hidden layers and connected inputs and outputs) that should facilitate the evolutionary search. Especially in the initial generations ES-HyperNEAT runs the risk of being trapped in local optima where high fitness values can be achieved only by incorporating a subset of the available inputs. The idea of seeding with a bias towards certain types of structures is important because it provides a mechanism for emulating key biases in the natural world that are provided ultimately by physics. For example, evolution could be seeded with an ANN topology that resembles the organization of the cortical columns found in the human brain (Sporns, 2002), potentially allowing higher cognitive tasks to be solved.

Providing such bias means escaping the black box of evolutionary optimization to provide a kind of general domain knowledge. Even though ES-HyperNEAT could in principle discover

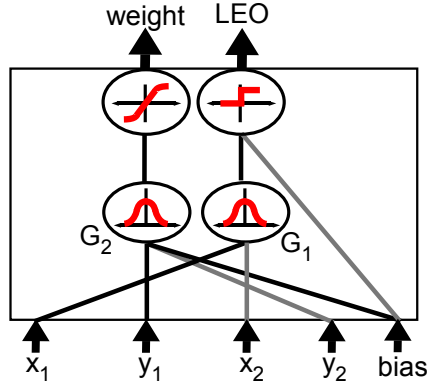


Figure 5.13: **X-locality and Geometry Seeding.** The CPPN is initialized with two Gaussian hidden nodes G_1 and G_2 that take as input $x_1 - x_2$ and $y_1 - y_2 + b$, respectively. Whereas G_1 is connected to the LEO with a bias of -1, G_2 connects to the weight output. G_1 peaks when x_1 and x_2 are the same, thereby seeding the initial CPPNs with locality along the x -axis. G_2 creates horizontal stripes of differing weights running through the hypercube, which induces the expression of multiple hidden layers in the decoded ANN. Positive connections are dark whereas negative connections are light.

the appropriate ANN topology by itself, biasing the search with a good initial topology should thus make the search less susceptible to local optima. While ES-HyperNEAT can modify and elaborate on such initial ANN topologies, FS-HyperNEAT would likely not benefit from geometry seeding because it cannot compensate for movement of information within the hypercube and certain structures are a priori not possible to represent if the nodes are not placed in the correct locations. Figure 5.13 shows a CPPN that combines seeded locality *and* seeded geometry. In addition to Gaussian node G_1 that specifies locality along the x -axis (Verbancsics and Stanley, 2011), a second Gaussian node G_2 is added that receives $y_1 - y_2 + b$, where b is bias, as input (figure 5.13) and therefore creates horizontal stripes of differing weights running through the hypercube along y_1 . Interestingly, changing the bias input of G_2 thus can immediately create ANNs with more or less *hidden layers*. In

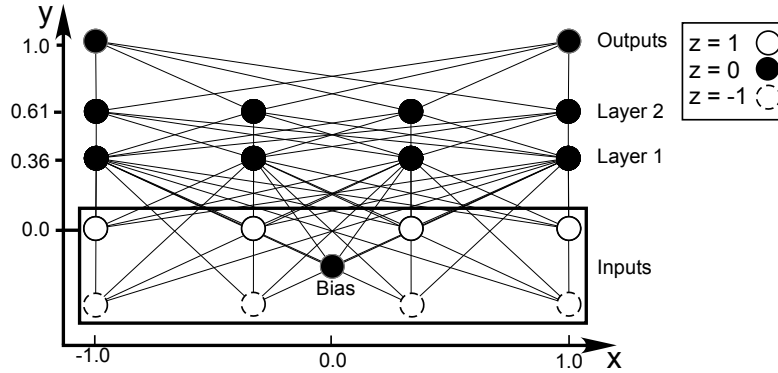


Figure 5.14: **Retina Problem Substrate Configuration.** The substrate consists of four layers with the inputs at $y = 0.0$, the first hidden layer at $y = 0.37$, a second hidden layer at $y = 0.61$ and the outputs at $y = 1.0$. Note that this substrate has three dimensions. The z coordinates are indicated by the different circle patterns.

the experiments reported here a bias weight of 0.33 was chosen, resulting in initial ANNs with two hidden layers, which is similar to the setup of the fixed-substrate variant explained in the next section. It is important to note that most connections will initially not be expressed because of the locality seeding. However, slight perturbations of the seed in the initial generation provide a variety of local connectivity patterns and ANN topologies. That the organization of a locally-connected two-hidden-layer network can be entirely described by two new hidden nodes in the initial CPPN suggests the power of the encoding and the expressiveness that is possible when seeding ES-HyperNEAT.

5.7.3 Retina Problem Setup

This substrate configuration is derived from the setups of Verbancsics and Stanley (2011) and Clune et al. (2010), which established such a three-dimensional setup as standard for the retina problem (figure 5.14). The ANN has eight input nodes and two output nodes. Fixed-substrate HyperNEAT has two layers of hidden nodes, with four hidden nodes each. Note that the substrate has three dimensions (x, y, z) . The ANN inputs receive either -3.0 or 3.0 depending on the state (e.g. off or on) for each retina input. The left and right outputs specify the classification for the left and right retinas, respectively, where values close to 1.0 indicate valid patterns. Values close to -1.0 indicate an invalid pattern. Six different HyperNEAT approaches are compared to isolate the effects of the LEO and the hidden layer seeding:

- In the **ES-HyperNEAT** approach the placement and density of the hidden nodes and their connections to the input and output nodes are determined entirely from the CPPN by the algorithm in Section 5.3 without any seeding.
- **ES-HyperNEAT-LEO** extends ES-HyperNEAT with LEO, which should facilitate the evolution of modular ANNS.
- **ES-HyperNEAT with Geometry Seeding** tests ES-HyperNEAT’s ability to take advantage of initial geometric seeding through the CPPN. The seed, shown in figure 5.13, creates ANNs with two hidden layers with four neurons each, corresponding

to the fixed-substrate retina setup (figure 5.14). Note that the functionality of the LEO is disabled in this setup.

- **ES-HyperNEAT-LEO with Geometry Seeding** tests the hypothesis that both extensions, LEO with locality seeding and geometric seeding (figure 5.13), should be complementary in increasing ES-HyperNEAT’s ability to generate modular networks for complicated classification problems.
- Following Verbancsics and Stanley (2011), **FS-HyperNEAT-LEO with x -locality seeding** is the original HyperNEAT approach with a fixed substrate and an additional LEO.
- In the **FS-HyperNEAT-LEO with Geometry Seeding** approach FS-HyperNEAT is also seeded with x -locality and initial weight patterns (figure 5.13) that intersect the positions of the fixed hidden nodes (figure 5.14). The hypothesis is that the additional geometric seeding should not significantly increase FS-HyperNEAT’s performance because small variations in the initial seed will disrupt the alignment between the CPPN-expressed pattern and hidden node positions, for which FS-HyperNEAT cannot compensate.

Note that the seed CPPN for all approaches without geometry seeding does not have direct connections from the inputs to the weight output in the CPPN. However, mutations on the seed to create the initial generation can connect arbitrary inputs to arbitrary outputs. This

setup has shown a generally better performance than starting with a fully-connected CPPN. Fitness, which is computed the same way for all approaches, is inversely proportional to the summed distance of the outputs from the correct values for all 256 possible patterns:

$$F = \frac{1000.0}{(1.0 + E^2)}, \text{ where } E \text{ is error.}$$

5.7.4 Results

All results are averaged over 40 runs. Figure 5.15 shows the training performance over generations for the different HyperNEAT variants and how frequently they solve the problem perfectly (i.e. correctly classify 100% of the patterns). Because FS-HyperNEAT-LEO succeeds in almost every run after 5,000 generations (Verbancsics and Stanley, 2011), to highlight the differences in the HyperNEAT variants, figure 5.15b reports the number of successful runs after a much shorter period of 2,000 generations.

While ES-HyperNEAT solves the retina domain in only 30% of the runs, augmenting ES-HyperNEAT with LEO and geometry seeding improves the odds of finding a solution to 57%. ES-HyperNEAT-LEO with geometry seeding performs significantly better than the other variants ($p < 0.05$), confirming Hypothesis 4 and the advantages of both ES-HyperNEAT extensions for the evolution of modular ANNs. The performance of the original FS-HyperNEAT with LEO, on the other hand, even decreases from finding a solution in 30% of the runs to

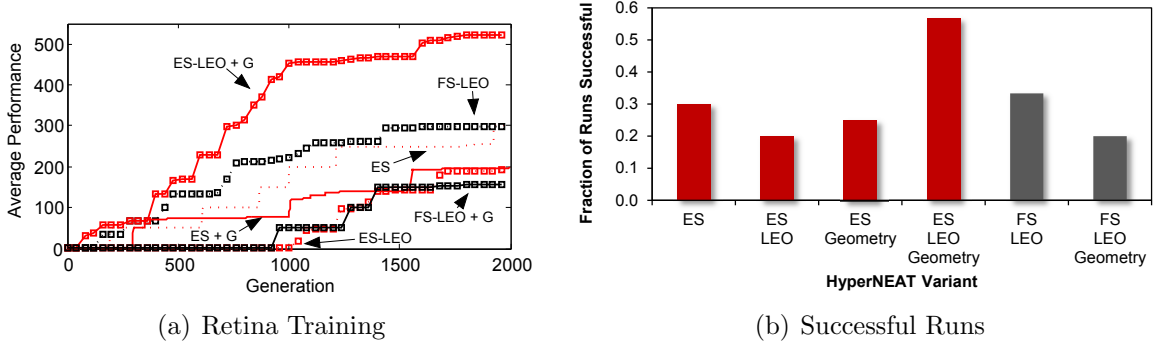


Figure 5.15: **Average Performance in Retina Domain.** The average best fitness over generations is shown for the retina domain (a) for the different HyperNEAT variants, which are averaged over 40 runs. The fraction of 40 runs that successfully solve the domain is shown in (b) for each of the HyperNEAT variants after 2,000 generations. The main result is that ES-HyperNEAT-LEO with geometry seeding significantly outperforms all other approaches.

only finding the solution in 25%, when seeded with geometry, confirming that only ES-HyperNEAT can take advantage of such geometry seeding.

Surprisingly, just extending ES-HyperNEAT with a LEO *alone* does not increase performance but instead decreases it slightly (though not significantly). Especially in the first 1,000 generations there is almost no improvement in performance (figure 5.15a), which suggests that pruning connections based on the amount of information in the hypercube and additionally through the LEO *without* any geometry seeding hinders the evolution of functional networks (i.e. ANNs with paths from the input to the output neurons). Additionally, only seeding with geometry but not extending ES-HyperNEAT with a LEO (ES-HyperNEAT with geometry seeding) also decreases performance, which is likely due to the increased crosstalk in the more fully-connected ANNs.

A closer look at the structure of some final solutions gives insight into how ES-HyperNEAT-LEO can elaborate on initial geometric seeding (figure 5.16). ES-HyperNEAT-LEO can successfully build on the initial structure, creating networks of varying complexity and resemblance to the initial seed (figure 5.13). Modularity is the prevailing pattern (figure 5.16a,b,e) but non-modular ANNs also emerge (figure 5.16c). While most networks display a high degree of symmetry (figure 5.16a-c), resembling the symmetry in the retina patterns (figure 5.12), less symmetric ANNs are also discovered (figure 5.16d,e).

The main result is that it is the combination of LEO and geometry seeding that allows ES-HyperNEAT more easily to evolve modular ANNs. The original HyperNEAT approach, on the other hand, cannot take full advantage of those extensions and performs significantly worse, even though it too benefits from the LEO.

5.8 ES-HyperNEAT Implications

The central insight in this chapter is that a representation that encodes the pattern of connectivity across a network (such as in HyperNEAT) automatically contains implicit clues on where the nodes should be placed to best capture the information stored in the connectivity pattern. Experimental results show that ES-HyperNEAT significantly outperforms the original HyperNEAT while taking a step towards more biologically plausible ANNs and sig-

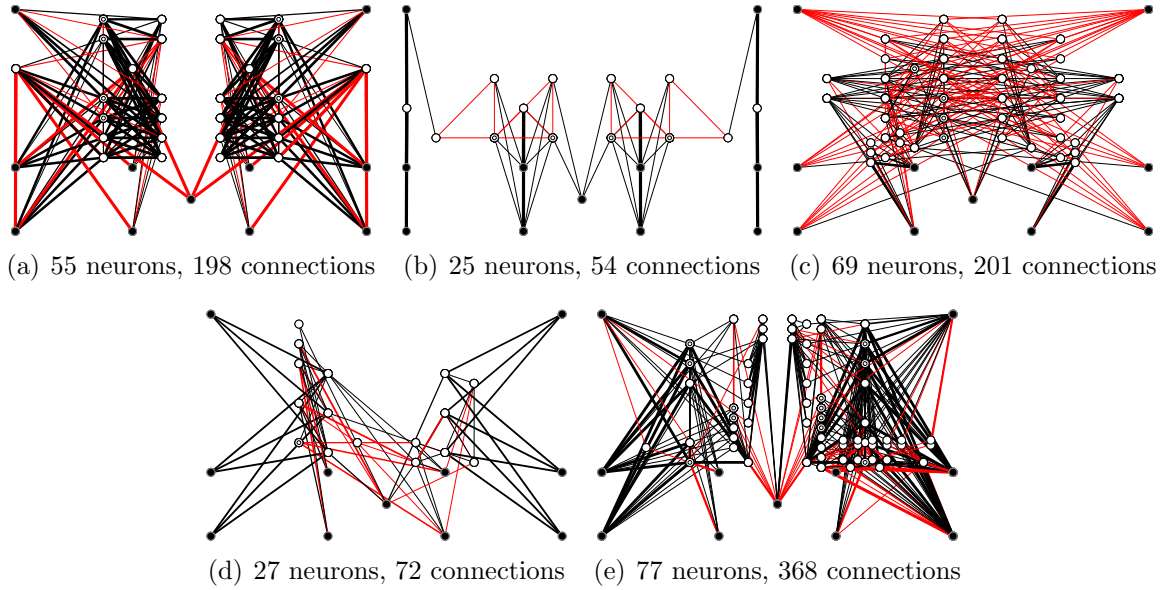


Figure 5.16: Example Connectivity Patterns from ES-HyperNEAT-LEO with Geometry Seeding in a Modular Domain. Modularity is commonly found when ES-HyperNEAT-LEO is seeded with the geometry seed (a, b). Once modularity is found, the regularities needed to solve the task for each module can be discovered in the weight pattern. ES-HyperNEAT-LEO evolves a variety of different ANNs with more or less hidden neurons and connections. Non-modular ANNs that solve the task are also discovered (c), although with less frequency. Hidden nodes with recurrent connections are denoted by a smaller concentric circle. Positive connections are dark (black) whereas negative connections are light (red).

nificantly expanding the scope of neural structures that evolution can discover. This section explores the implications of this capability and its underlying methodology.

5.8.1 Dictating Node Locations

The convention in HyperNEAT of the last several years that the user would simply decide a priori where the nodes belong evaded the deeper mystery about how connectivity relates to node placement. As suggested by Hypothesis 2, dictating the location of nodes makes it harder for the original HyperNEAT to represent the correct pattern, which the reported results in a variety of domains confirm. While ES-HyperNEAT can compensate for movement of information within the hypercube by expressing the hidden nodes at slightly different locations (e.g. figure 5.10c,d), representing the correct pattern for the original HyperNEAT is more difficult, resulting in more complex CPPNs. The significantly reduced performance of the vertical node arrangement FS1x10 in the maze navigation domain (figure 5.9a) indicates that the more complex the domain the more restrictive it is to have nodes at fixed locations.

One way to interpret the preceding argument is that the locations of useful information in the hypercube are where the nodes need to be. That way, the *size* of the brain is roughly correlated to its *complexity*. There is no *need* for a billion neurons to express a simple Braitenberg vehicle. Even if a billion neurons were summoned for the task, many of their functions would end up redundant, which geometrically means that large cross-sections of

the hypercube would be uniform, containing no useful information. The ES-HyperNEAT approach presented here is a heuristic attempt to formalize this notion and thereby correlate size to complexity. In this context, nodes become a kind of harbinger of complexity, proliferating where it is present and receding where it is not. Thus the solution to the mystery of the relationship between nodes and connections is that nodes are sentinels of complex connectivity; they are beacons of information in an infinite cloud of possible connections.

5.8.2 Incrementally Building on Stepping Stones

Previous work showed that ES-HyperNEAT and the original HyperNEAT exhibit similar performance in a simple navigation domain (Risi et al., 2010b). However, in the more complicated navigation domain presented here (Section 5.6), the best fixed-substrate HyperNEAT method (FS10x1) solves the domain in only 45% of runs. How can this poor performance be explained?

The problem is that the increased complexity of the domain requires incrementally building on previously discovered stepping stones. While direct encodings like NEAT (Stanley and Miikkulainen, 2002, 2004) can complexify ANNs over generations by adding new nodes and connections through mutation, the indirect HyperNEAT encoding tends to start already with fully-connected ANNs (Clune et al., 2010), which take the entire set of ANN connection weights to represent a partial task solution. On the other hand, ES-HyperNEAT is able

to elaborate on existing structure in the substrate during evolution (figure 5.10), confirming Hypothesis 3. This result is important because the more complicated the task, the more likely it will require a neuroevolution method that benefits from previously-discovered stepping stones.

These results also explain why uniformly increasing the number of hidden nodes in the substrate does not necessarily increase HyperNEAT’s performance. In fact, FS8x8 performs significantly worse than FS5x5, which is likely due to the increased crosstalk that each neuron experiences.

The ES-HyperNEAT solutions are in general more complex than necessary for the navigation task. Yet they are nevertheless optimized more quickly than their fixed substrate counterparts. An indirectly-encoded neuroevolution algorithm is not the same as a direct encoding like NEAT that complexifies by adding one node at a time to find just the right number of nodes for a task. The promise of the indirect encoding is rather to evolve *very large* networks that would be prohibitive to such direct encodings, with thousands or more nodes. Figure 5.9c shows that the substrate evolution algorithm can actually create increasingly complex indirectly-encoded networks even though it is not explicitly designed to do so (e.g. like regular NEAT) and that it can encode large ANNs from compact CPPN representations. One reason for this capability is that the impact of adding new nodes and connections to the CPPN is that the complexity of the pattern it encodes will often increase. Because ES-HyperNEAT in effect attempts to match the structures it discovers in the hypercube to the

complexity of the pattern within it, it makes sense that as the size of the CPPN increases, the complexity of the ANN substrate it encodes would generally increase as well.

So far NE techniques have mainly focused on evolving behaviors that require relatively small ANNs. ES-HyperNEAT’s new capabilities and its higher evolvability (figure 5.11) should enable more complex tasks to be solved in the future that require large-scale unknown node placements and the traversal of many stepping stones. The more complex the task, the more important it will be to free the user from the burden of configuring the substrate by hand. Also importantly, the approach has the potential to create networks from several dozen nodes up to several million, which will be necessary in the future to create truly intelligent systems.

5.8.3 Geometry Seeding

Whereas the original HyperNEAT only allowed seeding with a bias towards local connectivity (Verbancsics and Stanley, 2011), ES-HyperNEAT can also be seeded with a bias towards certain ANN topologies. The results in the retina problem confirm Hypothesis 4: the combination of LEO and geometry seeding allows the approach more easily to evolve modular ANNs, yet the original HyperNEAT approach cannot take full advantage of such a seed.

While the work of Verbancsics and Stanley (2011) took a step in the direction of incorporating important geometric principles (e.g. locality) that are helpful to create structures that

resembles those of nature, the geometry seeding presented here goes a step further. The idea of seeding with a bias towards certain types of structures is important because it provides a mechanism for emulating key biases in the natural world that are provided ultimately by physics.

As noted by Verbancsics and Stanley (2011), the better performance with an initial bias towards locality suggests that the path to encoding locality is inherently deceptive with respect to the fitness function in the retina task. Such deception may turn out common when geometric principles such as locality that are conceptually orthogonal to the main objective are nevertheless essential to achieving the goal. Thus seeding with the right geometric bias may prove an important tool to avert deception in many domains.

Overall, ES-HyperNEAT thus advances the state-of-the-art in NE beyond the original HyperNEAT and has the potential to create large-scale ANNs for more complicated tasks, like robots driven by raw high-resolution vision, strategic game-players, or human assistants. For the field of AI, the idea that we are beginning to be able to reproduce some of the phenomena produced through natural evolution (e.g. compactly encoded regular and modular networks) at a high level of abstraction is important because the evolution of brains ultimately produced the seat of intelligence in nature.

The next chapter augments ES-HyperNEAT with adaptive HyperNEAT’s ability to indirectly encode plastic ANNs as a pattern of local learning rules, which yields a unified approach that can evolve agents for more complex task that require lifetime learning.

CHAPTER 6

UNIFIED APPROACH: ADAPTIVE ES-HYPERNEAT

To date no method in neuroevolution unifies the ability to indirectly encode connectivity through geometry, simultaneously encode the density and placement of nodes in space, and generate patterns of heterogeneous plasticity. While such a unified approach risks being ad hoc, HyperNEAT offers a unifying principle that naturally expresses both node placement and plasticity. That is, both can be encoded by the CPPN as a pattern situated within the geometry of the substrate.

Thus this chapter introduces for the first time the complete *adaptive ES-HyperNEAT* approach that can evolve increasingly complex neural geometry, density, and plasticity. The experiments in this chapter are to appear in Risi and Stanley (2012).

6.1 Evolving Plasticity and Neural Geometry

The Hebbian ABC model (Section 4.1) was chosen as the basis for extending ES-HyperNEAT to also evolve plastic ANNs because it offers a good compromise between the generality of an indirect encoding of plasticity and its computational cost. Additionally, given the right topology (which ES-HyperNEAT should be able to evolve), the ABC model should be sufficient for most domains.

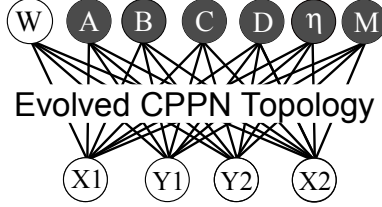


Figure 6.1: **Adaptive ES-HyperNEAT CPPN.** This CPPN is only activated once for each queried connection to determine its weight and learning parameters. This model completely determines weights, plasticity, and modulation through a single vector function encoded by the CPPN. Node placement is further determined by implicit information in the pattern within the hypercube, following the ES-HyperNEAT approach.

Adaptive ES-HyperNEAT augments the four-dimensional CPPN that normally encodes connectivity patterns with six *additional* outputs beyond the usual weight output: learning rate η , correlation term A , presynaptic term B , postsynaptic term C , constant D , and modulation parameter M (figure 6.1). The pattern produced by parameter M encodes the modulatory connections, as explained below. When the CPPN is initially queried, these parameters are permanently stored for each connection, which allows the weights to be modified during the lifetime of the ANN. In addition to its standard activation, each neuron i also computes its modulatory activation m_i based on the types of its incoming connections: $m_i = \sum_{w_{ji} \in \text{Mod}} w_{ji} \cdot o_j$. The weight of a connection between neurons i and j then changes following the m_i -modulated plasticity rule: $\Delta w_{ji} = \tanh(m_i/2) \cdot \eta \cdot [A o_j o_i + B o_j + C o_i + D]$. That way, HyperNEAT allows connection-mediated neuromodulation (i.e. connections *instead* of neurons (unlike in Section 2.5.2) are either standard or neuromodulatory), because that way HyperNEAT can efficiently encode plasticity as a pattern of local rules.

The other important component of adaptive ES-HyperNEAT is that it determines the placement and density of nodes from implicit information in the hypercube encoded by the CPPN, as in ES-HyperNEAT alone (Chapter 5). Among the seven CPPN outputs, the node placement and connectivity is determined by searching only through the patterns produced by the weight output W (to discover regular connections) and modulatory output M (to discover modulatory connections). These patterns in effect represents a priori information at the “birth” of the ANN, which by convention then determines node placement. As noted earlier, the node-placement algorithm searches for areas of high variances within the hypercube, in effect giving the CPPN a *language* to express density patterns (Risi and Stanley, 2011). Once all hidden neurons are discovered, only those are kept that have a path to an input *and* output neuron.

Thus adaptive ES-HyperNEAT is able to fully determine the geometry, density, and plasticity of an evolving ANN based on a function of its geometry. While work in Chapter 5 showed the benefits of evolving the HyperNEAT substrate, the next section presents an experiment designed to demonstrate the advantages of augmenting the model with synaptic plasticity.

6.2 Continuous T-Maze Domain

The domain investigated in this chapter is a more complicated version of the T-Maze task studied in Chapters 3 and 4. The most important aspect of this domain is that it is not the

traditional discrete grid-world T-Maze that is popular in this area (Soltoggio et al., 2008); rather, the more realistic domain presented here (figure 6.2) requires the robot to develop both collision avoidance and the ability to learn during its lifetime in a continuous world (unlike the traditional discrete T-Maze). Similar experiments with neuromodulated ANNs were performed by Dürr et al. (2010) but their robot relied on additional ANN inputs (e.g. an input that signaled the end of the maze and an input that was activated once the T-Maze junction came into view) to solve the task. Additionally, to facilitate the task, their robots had only two infrared sensors whose values were merged into one sensory input. The increased number of rangefinder inputs (five) in the present experiment gives the simulated robot a finer resolution without the need for any sensor preprocessing. Blynal and Floreano (2003) also solved a version of the continuous task, but with CTRNNs instead of plastic networks, which often required incremental evolution. Therefore, this challenging version of the T-Maze domain should test adaptive ES-HyperNEAT’s ability to directly evolve more sophisticated plastic ANNs.

To generate a controller for the T-Maze domain, the evolved CPPNs query the substrate shown in figure 6.2b. The placement of input and output nodes on the substrate is designed to geometrically correlate sensors and effectors (e.g. seeing something on the left and turning left). Thus the CPPN can exploit the geometry of the robot. The simulated robot is equipped with five rangefinder sensors that are scaled into the range $[0, 1]$. The *Reward* input remains zero during navigation and is set to the amount of reward collected at the maze end (e.g. 0.1 or 1.0). At each discrete moment of time, the number of units moved by the robot is $20F$,

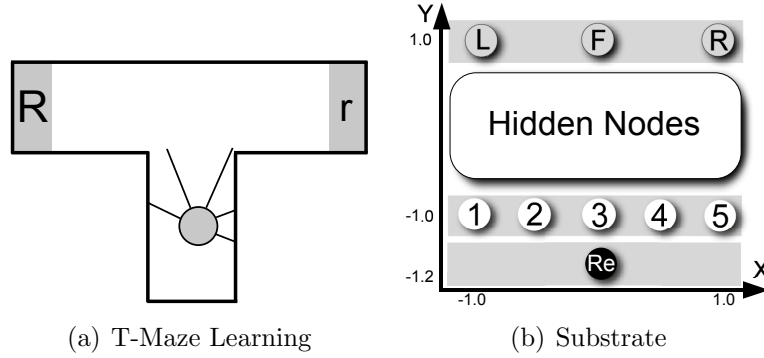


Figure 6.2: **Continues T-Maze Domain and Substrate Configuration.** (a) The challenge for the robot in the T-Maze domain is to remember the location of the high reward from one trial to the next. (b) The simulated robot is equipped with five distance sensors (white) and one reward sensor (black).

where F is the forward effector output. The robot also turns by $17(R - L)$ degrees, where R is the right effector output and L is the left effector output.

The goal of the robot is to maximize the amount of reward collected over four independent deployments with varying initial reward positions and switching times. Because Chapter 3 showed that only measuring performance based on the amount of collected reward is not a good indicator of the adaptive capabilities of the evolved agent, the fitness function in this chapter rewards consistently collecting the same reward, with a score of 1.0 for each low reward and a score of 2.0 for each high reward collected in sequence. The design of this fitness function is based on insights gained by analyzing typical behaviors archived by novelty search in the discrete T-Maze domain (figure 3.7). Experimental parameters for this experiment are given in the Appendix.

6.3 Continues T-Maze Results

Whereas ES-HyperNEAT alone could only solve the T-Maze task in one out of 30 runs (by storing state information through recurrent connections), adaptive ES-HyperNEAT found a solution in 19 out of 30 runs, in 426 generations on average ($\sigma = 257$) when successful (figure 6.3a). This result suggests that augmenting ES-HyperNEAT with the ability also to encode plastic ANNs is important for tasks that require the agent to adapt. Evolved solutions always collect the maximum-possible reward without colliding and only require a minimum amount of exploration (i.e. collecting a low reward) at the beginning of their lifetime and when the reward changes position. Unlike in Dürr et al. (2010), the ANNs have no special sensors to designate key locations and they also take raw rangefinder input.

Figure 6.3b also shows that the neural dynamics start to resemble particular dynamics found in nature. Similarly to cells in the rat’s hippocampus (Eichenbaum, 2000), individual hidden neurons in a typical ANN solution only fire when the robot is in a left or right-side trial, indicating an episodic-like coding of events (i.e. each trial is encoded separately).

Figure 6.4 shows an example ANN solution and its underlying CPPN. The ANN has 149 connections and 32 hidden nodes. With seven parameters per connection, the network has a total of 1,043 parameters. In contrast, the CPPN that encodes this network has only 54 connections and 6 hidden nodes; thus it is 19 times smaller than the network it encodes. In this way, HyperNEAT is able to explore a significantly smaller search space (i.e.

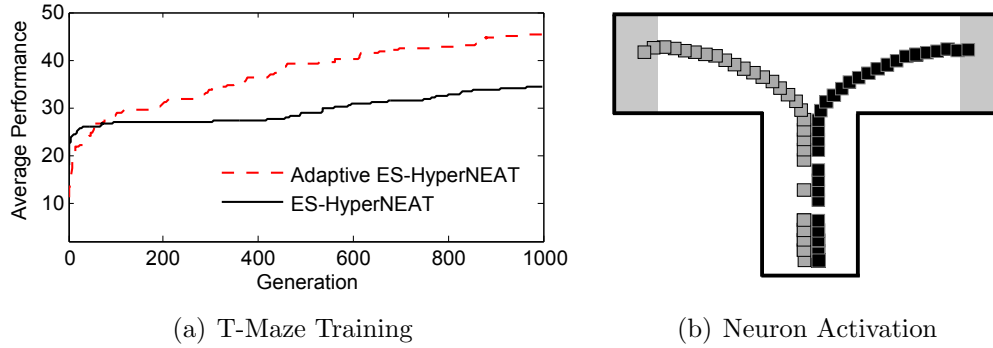


Figure 6.3: **Average Performance in Continues T-Maze.** The average best fitness over generations is shown for the T-Maze domain (a). The main result is that adaptive ES-HyperNEAT significantly outperforms ES-HyperNEAT because it allows the robot to more easily adapt during its lifetime. The activation of two hidden neurons from a T-Maze solution ANN is shown in (b). Only on trials with the high reward to the right does one of the two neurons show a positive activation (black) while the robot passes through the maze and turns onto the right choice arm. The other neuron only shows a positive activation (gray) on trials with the reward to the left, indicating an episodic-like encoding of events (Eichenbaum, 2000).

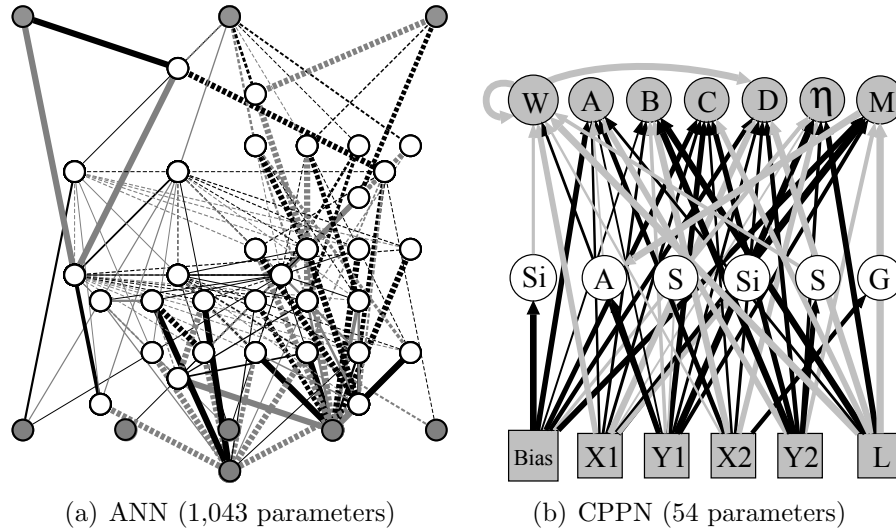


Figure 6.4: **Example Solution ANN Created by Adaptive ES-HyperNEAT and its Underlying CPPN.** Positive connections are dark whereas negative connections are light. Modulatory connections are dotted and line width corresponds to connection strength. The algorithm discovered this ANN solution (a) by extracting the information inherent in the simpler underlying CPPN (b). CPPN activation functions are denoted by *G* for *Gaussian*, *S* for *sigmoid*, *Si* for *sine*, and *A* for *absolute value*. The CPPN receives the length of the queried connection *L* as an additional input.

CPPNs) while still creating complex structures (i.e. substrates). The ANN includes both neuromodulatory and plastic connections, showing that the full heterogeneous plasticity of an adaptive network can be encoded by a single compact CPPN. Different regions of the substrate show varying levels of node density and the majority of the modulatory connections originate from the right side of the substrate. Similar neural structures can also be found in the brain, where groups of neurons, called diffuse modulatory systems, project to numerous other areas (Bear et al., 2007).

6.4 Adaptive ES-HyperNEAT Implications

The main results in this chapter signify progress in the field of neuroevolution. Evolved ANNs are beginning to assume more natural properties such as topography, regularity, heterogeneous plasticity, and neuromodulation. Furthermore, the results in the continuous T-Maze demonstrate that these capabilities combine effectively in a task that requires adaptation. Perhaps the most important point is that progress in this area has been cumulative: rather than separate methods that depend on entirely different architectures, the techniques combined in this study are built one upon another. In effect, a new research trajectory is coming into focus. For the field of AI, the idea that we are beginning to be able to reproduce some of the phenomena produced through natural evolution at a high level of abstraction is important because it might allow the creating of a new kind of intelligent machines.

The main idea behind adaptive ES-HyperNEAT is that the node placement and plasticity can be encoded by the CPPN as a pattern situated within the geometry of the substrate. The next chapter explores a variant of the idea that the CPPN can encode plasticity in a legged robot domain, in which the entire substrate is impacted by global morphological information rather than local neural activation. In this spirit, the ability of CPPNS to encode particular weight configurations based on external conditions is exploited and the network is able to *adapt* to a variety of different robot body proportions.

CHAPTER 7

TOWARDS AN ADAPTIVE NEUROCONTROLLER FOR LEGGED LOCOMOTION

Legged robots have long captured the interest of researchers in robotics (Hornby et al., 2000; Kimura et al., 2001; Wettergreen and Thorpe, 1992; Lewis and Bekey, 2002). Unlike their wheeled counterparts, they offer a high degree of mobility and excel on rugged terrain. However, designing a controller for a legged robot is challenging because of the high number of degrees of freedom within each leg and the need for tight coordination and balance.

Controllers for legged robots are often crafted and tuned for a specific robot morphology, which can be a difficult and time-consuming task (Wettergreen and Thorpe, 1992). Therefore, interest has increased in recent years in *automatically* learning gaits for particular robots. In one seminal work, Hornby et al. (2000) trained dynamic gaits (i.e. gaits that require transient instability during ambulation) for the AIBO robot dog with an evolutionary technique. The trained controller was actually included in the commercial release of the robot. In fact, a range of techniques have shown promise (Kimura et al., 2001; Lewis and Bekey, 2002; Clune et al., 2009).

However, learned and hand-designed controllers both are likely to fail when the robot's morphology is altered, even if only slightly (Hornby et al., 2000). This brittleness contrasts starkly with the capabilities of legged animals in nature. For example, a foal can quickly learn to walk a short time after birth. As noted by Lewis and Bekey (2002), it is unlikely that the

locomotion controller in the foal is determined completely before birth or that any “*learning algorithm could program a nervous system ab initio with so few training epochs*”. Thus rather than learning a controller for a particular morphology, an intriguing unexplored possibility is instead to learn a *relationship* between the morphology of a robot and its control. In fact, the ability to exploit such a relationship is a major goal for the field of robotics.

In this spirit, the ability of CPPNs to encode particular weight configurations based on external conditions is exploited not only to create ANNs as a function of the domain geometry but also based on the particular quadruped *morphology*. By evolving a function that can output controllers for different morphologies (figure 7.1), HyperNEAT can actually learn the relationship between morphology and control policy. This approach can be viewed as a variant of the idea behind adaptive HyperNEAT, except that in this realization, the entire substrate is impacted by global morphological information rather than local activation. That way, the network *adapts* to the robot’s size. The hope for such an approach, which has not been demonstrated previously, is that once such a relationship is learned the output controllers are able to work on a diversity of different morphologies *without further training*.

While this approach differs from how animals and humans gradually learn to adapt to their bodies of origin, it should provide useful conceptual insights towards the general goal of adaptive controllers for locomotion. To understand how a diversity of morphologies will be encoded by the same CPPN, it helps to begin by considering how a single CPPN encodes a single quadruped controller.

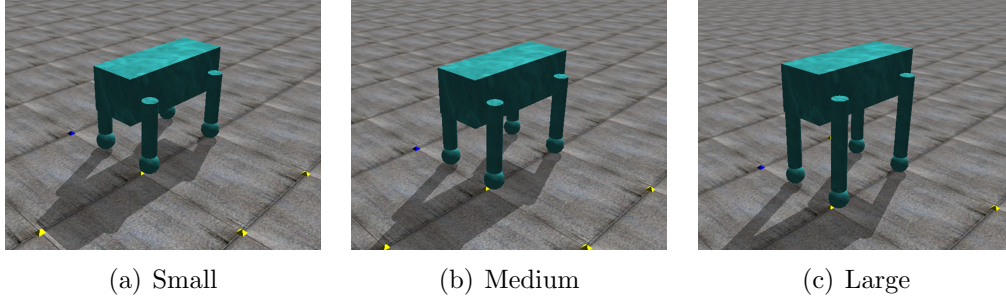


Figure 7.1: **Quadruped Training Morphologies.** The flexible neural controller trained on three quadrupeds with different leg segment lengths (0.25, 0.30, and 0.37 meters) learns to interpolate to never-seen intermediate morphologies without further training.

7.1 Static Quadruped Neural Controller

In this chapter, a three-dimensional quadruped robot (figure 7.1) in a realistic physics simulation using the freely available Open Dynamics Engine (see <http://www.ode.org>), is controlled by a *continuous-time recurrent neural network* (CTRNN) (Section 2.1) that is able to express the non-linear dynamics found in natural gaits and is common in other legged robot experiments (Reil and Husbands, 2002; McHale and Husbands, 2004). The quadruped robot (figure 7.1) has a total of twelve degrees of freedom (DOF): two degrees in each hip joint (pitch and roll) and one degree in each knee joint (pitch).

The neural architecture for the quadruped is distributed into separate substrate modules for each of the four legs (figure 7.2). Similar decentralized architectures seem critical in walking biological organisms (Pearson et al., 1976), and also have inspired hand-designed control architectures for hexapod robots (Brooks, 1989).

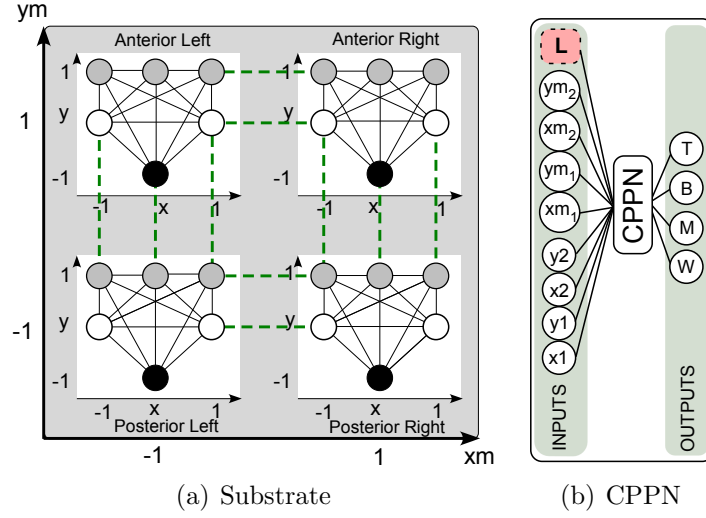


Figure 7.2: **Distributed Substrate Architecture and CPPN.** The quadruped architecture is divided into four different substrate modules (a). Each module is responsible for controlling a single leg, where the inter-module connections allow neighboring nodes in different modules to communicate (dotted line). Each leg module has one input (black), two hidden (white), and three output nodes (gray). The CPPN (b) takes the internal position of each node inside their corresponding module ($x.y$) and the the position of the module itself (x_m, y_m) as input to generate the intra-module connections (determined by output W), inter-module connections (output M), time constants (output T), and node biases (output B). The flexible neural controller additionally receive the current leg length L as input when determining the time constant and bias values of the hidden nodes.

Each leg module has one input, two hidden, and three output nodes. The inputs provide the current angle of the anterior-posterior hip joint. The ANN outputs movement requests for each degree of freedom in the model, i.e. for each independent axis of rotation for all joints in the model. The outputs are scaled to match the angular range of the corresponding DOF, which is interpreted as the angle that the neural controller is requesting. The difference between the requested angle and the current orientation of the DOF denotes the disparity between the state the ANN is requesting and the current state of the model. A proportional controller applies torque to reduce this disparity. In other words, the ANN directs the low-level controllers towards a particular state. This method of control are similar to those in Reil and Husbands (2002) and Lehman and Stanley (2011b).

To generate a controller for the quadruped, a CPPN (figure 7.2b) with inputs $x_1, y_1, x_2, y_2, x_{-m_1}, y_{-m_1}, x_{-m_2},$ and y_{-m_2} first queries each of the four substrate modules shown in figure 7.2a to determine the intra-module connection weights between the input/hidden, hidden/hidden, hidden/output, and output/output nodes of each module (determined by CPPN output W). That is, inputs x and y describe the internal position of each node inside their corresponding module, whereas the x_{-m} and y_{-m} inputs determine the position of the module itself (e.g. the module controlling the anterior left leg is located at ($x_{-m} = -1, y_{-m} = 1$)) within the larger substrate. The CPPN can thus emphasize the influence of x_{-m} and y_{-m} for increasing heterogeneity or minimize it to produce greater homogeneity between different leg modules. Subsequently, the inter-module connections between corresponding neighboring hidden and output neurons are determined (dotted lines in figure 7.2a) by CPPN output

M . This approach to encoding separate modules within a single substrate is inspired by the multiagent HyperNEAT approach (D’Ambrosio et al., 2010), which showed how several similar ANNs can be encoded by the same CPPN. However, an added idea here is also to encode connections *between* such modules, to help them synchronize in time.

Additionally the bias and time constant values for each *hidden* node are determined by CPPN outputs T and B . By convention those values are determined by a node-centric query at $(x_1, y_1, x_{-m_1}, y_{-m_1}, 0, 0, 0, 0)$, where $x_2, y_2, x_{-m_2}, y_{-m_2}$ are simply set to zero. The time constants and bias values for the *output* nodes are fixed for all approaches to 0.1 and 3.0, respectively. These values have been found to work well for a variety of different morphologies. Most importantly, the key idea in this section is that the CPPN takes a set of geometric parameters and outputs a connectivity pattern for a network.

7.2 Flexible Neural Controller

A flexible network controller poses a greater challenge; how can a single CPPN encode a set of different architectures for different quadruped morphologies, all related but requiring slightly different gaits?

The main idea is that a single CPPN is able to create different ANNs based on both the robot’s internal geometry and its current leg length. While the connectivity pattern of the

network is generated as described in the previous section, now the *leg length* is provided as an additional input to the CPPN for the node-centric queries of the time constants and biases of the hidden nodes. These hidden nodes function as the main Central Pattern Generators (CPGs) of the quadruped, controlling the rhythmic movement of the legs of the robot. CPGs have been suggested as important functional units of the central nervous system (Grillner and Wallen, 1985). The hypothesis in this work is that given the right neural connectivity, a change in the time constants can modulate the gait of the robot to fit a certain morphology. Additionally, by supplying the CPPN with a morphology-dependent input (e.g. leg length) and evaluating it on a variety of different morphologies, it should be able to learn a relationship between morphology and control. In effect the CPPN encodes an adaptive controller dependent upon leg length.

While the CPPN in this chapter is only augmented with a single additional input, in principle more inputs could be added (i.e. joint ranges, maximum torque, torso densities, etc.), allowing the CPPN to produce an even wider range of different controllers for different robot morphologies.

A key question about such a flexible controller is whether the CPPN can also in principle generate controllers for intermediate leg scales on which it was not trained. This capability would allow one CPPN to produce ANNs for a variety of different morphologies without further training. The controllers would be able to interpolate between the policies of the

learned morphologies, thereby allowing less model-specific controllers than have heretofore been possible.

7.3 Experimental Setup

Because initial random controllers for legged robots and all of their immediate perturbations tend to fall they provide a bad gradient for an objective-driven evolutionary search (Van De Panne and Lamouret, 1995; Lehman and Stanley, 2011b). Therefore, the experiments in this chapter employ novelty search (Section 2.6), which has also been shown to overcome deception in locomotion-based problems in the past (Lehman and Stanley, 2011b). Chapter 3 also showed how novelty search supports learning plastic controllers. For the purposes of this study, it is the chosen learning approach for both static and flexible controllers. In the future, when more is known about the proper incentives to encourage flexible controllers to evolve consistently, a more targeted methodology may be warranted. However, at present novelty search, which is agnostic about the specific objective of the search, is a good choice for an initial exploration of what is possible.

To investigate the effect of evolving flexible controllers, both static and flexible neural architectures are trained and tested. Each neural controller is evaluated on three robot morphologies (figure 7.1) with leg segment lengths of 0.25, 0.30, and 0.37 meters. Each evaluation lasts for the duration of 15 simulated seconds and is terminated if the robot falls.

Creating controllers for robot morphologies not seen in training is challenging because the new robots must be assigned gaits automatically. Yet such a capability could be important to learning controllers for whole ranges of morphologies. To test this capability, the CPPNs recorded in novelty search’s permanent archive during training are tested on several different quadrupeds with intermediate leg sizes without further learning. Experimental parameters for this experiment are given in the Appendix.

7.4 Quadruped Results

To investigate the potential for flexible controllers, results were collected from 25 runs of static-controller evolution and 25 runs of flexible-controller evolution. The main question is whether the flexible neural controller’s ability to potentially learn a relationship between morphology and control benefits its performance when transferred to never-seen intermediate morphologies.

In training on the three morphologies shown in figure 7.1 both methods perform similarly. The final performance on these three cases with the flexible method is 4.88 meters traveled on average ($\sigma = 3.15$) in the allocated time, which is slightly (though not significantly according to the Student’s T-test) better than the static method, which travels 3.84 meters on average ($\sigma = 1.21$). In fact, this result simply reflects the fact that most runs of both approaches do not produce effective walkers on all three morphologies, bringing the average

down. However, a significant difference between the two methods is their standard deviation ($p < 0.0001$ according to the F-test). This result is interesting because it suggests that the *best* controllers possible are more likely to be found at the tail of the distribution of flexible controllers, which is where the potential for genuine interpolation may be confirmed.

Indeed, the *best gait* discovered by the dynamic method traveled 12.39 meters (averaged over the three training morphologies), while the best gait discovered by the static method traveled only 6.24 meters. Figure 7.3 shows the neural dynamics of the best evolved gait from the flexible setup for the medium sized robot (leg segment length = 0.30 meters). Evolution discovered a dynamic trot gait, which is characterized by the motor outputs of the opposing legs being in antiphase with each other.

7.4.1 Interpolation Performance

More interestingly, the key question is whether any such controllers can work on morphologies never seen in training. Recall that the flexible neural approach encodes multiple controllers as a function of their morphology. To test the capability to interpolate to unseen morphologies, leg segment increments of 0.01 meters in the interval from 0.25 (small training morphology) to 0.37 meters (large training morphology) were attempted. An interesting discovery is that two flexible controllers are able to pass the threshold of 6.0 meters in 12 and 11 out of 13 morphologies, respectively (figure 7.4). In contrast, the best static controller *fails* for 11 out

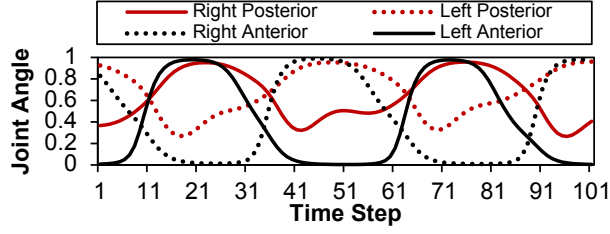


Figure 7.3: **Motor Neuron Dynamics.** The neural dynamics of the four hip outputs (pitch) controlling a medium quadruped with 0.30 meters leg segment length are shown. The dynamics of the opposing legs (e.g. anterior left and anterior right) are directly in antiphase with each other, which results in a dynamic trot-like gait.

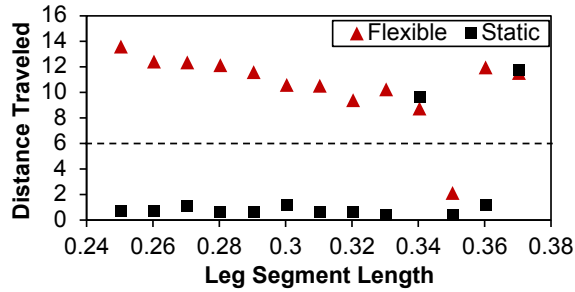


Figure 7.4: **Best Interpolation Performance.** The performance of the controller that interpolates best for both approaches is shown for 13 different morphologies. The flexible approach almost always interpolates perfectly (i.e. reaches the threshold value of 6.0 meters), while the best static controller fails in 11 out of 13 cases.

of the 13 morphologies. Reflecting the high variance in solutions, the flexible method solves on average 2.16 intermediate morphologies ($\sigma = 3.68$), while the static approach only solves 0.52 on average ($\sigma = 0.71$), which is statistically significant ($p < 0.05$). Videos of this best evolved controller are available at: <http://youtu.be/X-nQ0xgceJs>.

Thus the results do confirm that such relationships exist and can be discovered. The CPPN that outputs these controllers for different leg lengths is the first network interpolator of its kind of which we are aware off.

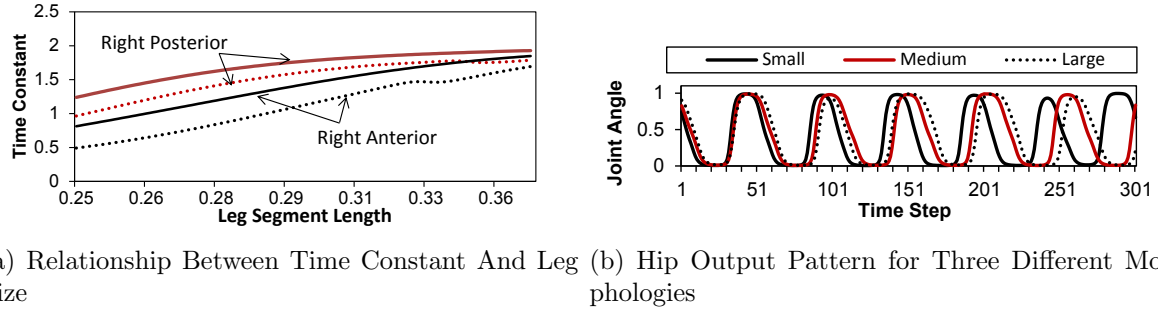


Figure 7.5: **Morphology-Dependent Change.** The relationship between the leg length of the robot and the time constant values of the hidden nodes are shown in (a). While the increase in leg length has almost no effect on the time constant values of the hidden nodes controlling the left side of the robot (data not shown), an increase in leg sizes correlates clearly with an increase of the time constant values controlling the two right legs. The right anterior hip (pitch) output generated by the three training quadrupeds are shown in (b). The period of the oscillation increases with the length of the robot’s legs, resulting in a prolonged leg stride.

7.4.2 Relationship Between Morphology and Control

The better performance of the flexible approach stems from its ability to generate a neural controller fitted to a specific morphology. How is this relationship represented in the underlying architecture? Figure 7.5a shows a correlation between time constant values of the hidden nodes, which are determined by the CPPN, and the change of the *leg length* CPPN input.

In particular, the time constant values of the hidden nodes controlling the right side of the robot exhibit a significant positive correlation ($r = 0.97$, $p < 0.01$) with the increased leg segment length. This change in time constant values results in an increased period of the hip (pitch) oscillation (figure 7.5b), thereby prolonging the leg stride of the robot (i.e. the larger

the robot’s legs, the larger the leg strides it has to make). This correlation suggests that the CPPN has actually learned a functional relationship between different robot morphologies and the network architectures that are necessary to control them.

7.5 Implications of Flexible Controllers

The major contribution of this chapter is conceptual. It introduces the idea that learning a relationship between morphology and control may ultimately be more practical than learning individual controllers. Yet to learn such a relationship requires a learning method that *outputs* controllers, which is what a CPPN can do. In this sense the CPPN is a powerful abstraction able to encode more than one kind of plasticity (i.e. not restricted to local learning rules).

While training a flexible CPPN-based controller-generator on three different morphologies might be expected to generate controllers on those three morphologies, the surprise in this chapter was the discovery that the best such CPPNs actually can generate intermediate controllers as well for morphologies never seen in training. This discovery suggests that there is a tangible relationship between morphology and controller architectures that can be discovered and exploited, at least with quadrupeds. While the reward system in the experiment (i.e. novelty search) was not set up explicitly to encourage such interpolation,

now that we know it is possible to do it, a further important task is to craft reward functions that indeed produce interpolating controllers consistently.

This task is more complex than it may sound; it is likely that simply rewarding several intermediate morphologies would present a deceptive landscape to the search algorithm, which is one reason novelty search is a good initial choice. However, now that such solutions are confirmed to exist, the prospect of devising a better reward scheme to find them consistently is more promising.

While this chapter established the benefits of learning a relationship between morphology and control, naive interpolation might sometimes fail and a controller that has to interpolate between different morphologies and different robot specific properties (e.g. joint ranges, maximum motor forces, etc.), will likely not always work directly. However, HyperNEAT’s abilities to evolve complex regular ANNs (Chapter 5) combined with adaptive HyperNEAT’s ability to generate plastic networks (Chapter 6) should allow such controllers to fine-tune their synaptic connections to the current quadruped model in the future. Adaptive ES-HyperNEAT could also allow robots to be more resilient to morphological changes that were not encountered during evolution.

Perhaps in the future it will be possible to vary the morphology of certain canonical types of robots such as quadrupeds or bipeds and simply pop in a generic controller-generator that seamlessly adapt to their bodies of origin. Such a capability could one day liberate roboticists

from the minutia of designing morphology-specific controllers to concentrate instead only on the best morphology for the job.

CHAPTER 8

DISCUSSION AND FUTURE WORK

This chapter begins with a discussion of the importance of being able to reproduce some of the phenomena produced through natural evolution at a high level of abstraction. The following section then contemplates the two HyperNEAT extensions and their unification into adaptive ES-HyperNEAT. The final section covers future possibilities created by the results in this dissertation.

8.1 Towards More Brain-Like ANNs

Intelligence in nature is the product of living brains, which are themselves the product of natural evolution. These brains are the very inspiration for the field of AI and for research on neural computation in particular. After all, our idea of emulating intelligence is ultimately inspired by the observation that it exists in nature. Yet despite this central role of evolved brains in the inspiration for AI, the idea of literally trying to recapitulate the evolutionary process that produced intelligence on Earth has received attention only in the small subfield of neuroevolution.

Yet the one motivation that neuroevolution offers unlike any other area in AI is to illuminate how it was possible for an unguided process to produce structures of astronomical complexity

that translate into intelligence in its highest form. To understand such a process, even incompletely, is to shed light on the very genesis of intelligence on Earth, which is essential to the mission of AI.

In fact, while neuroevolution began as little more than an optimization heuristic for evolving simple ANNs, the HyperNEAT method and the HyperNEAT extensions presented in this dissertation have begun to yield such insight into the necessary ingredients for more brain-like structures to evolve. For the field of AI, the idea that we are beginning to be able to reproduce some of the phenomena produced through natural evolution at a *high level of abstraction* is important because brains are the seat of intelligence in nature. To be able to begin to produce artifacts of evolution that exhibit some natural properties means that these artifacts and their evolution can ultimately be studied and deconstructed in their own right, similarly to neuroscience but without the inconvenience of flesh and blood.

8.2 HyperNEAT Extensions

Two extensions to the HyperNEAT method are introduced in this dissertation together with their unification. The first is adaptive HyperNEAT (Chapter 4), which allows not only generating patterns of weights across the connectivity of an ANN as a function of its geometry, but also patterns of learning rules. To demonstrate adaptive HyperNEAT, a discrete T-Maze task was selected (Section 4.2). The experiments revealed that adaptive

HyperNEAT can successfully indirectly encode arbitrary learning rules but also that there is a tradeoff between the generality of an indirect encoding of plasticity and its computational cost. Yet, as experiments in a variant of the T-Maze domain showed, in special cases, e.g. when the reward signature is nonlinear and the ANN topology is restricted, a most general encoding may be necessary.

Chapter 7 explored a variant of the idea that the CPPN can encode plasticity, in which the entire substrate is impacted by global morphological information rather than neural activation. This form of plasticity allowed a legged robot to *adapt* to a variety of different body proportions and interpolate to morphologies never seen in training. The ability of CPPNs to encode particular weight configurations based on external conditions hints at the powerful possibility of adaptation without the need for local interaction. Reproducing some of the learning abilities of real brains at a high level of abstraction could give insights into their essential properties without introducing unnecessary inefficiencies.

The second extension to HyperNEAT is evolvable-substrate HyperNEAT (Chapter 5). The key insight behind ES-HyperNEAT is that a representation that encodes the pattern of connectivity across a network (such as in HyperNEAT) automatically contains implicit clues on where the nodes should be placed to best capture the information stored in the connectivity pattern. The approach was demonstrated through multi-task (Section 5.5), maze navigation (Section 5.6), and modular retina domains (Section 5.7), revealing that the neural networks generated by this new approach assume more natural properties such as neural topography

and geometric regularity. Analysis of the results and the evolved ANNs showed that ES-HyperNEAT outperforms the original HyperNEAT because of ES-HyperNEAT’s ability to evolve ANNs with partial and targeted connectivity, elaborate on existing ANN structure, and to compensate for movement of information within the underlying hypercube. Additionally, ES-HyperNEAT can more easily evolve modular ANNs when biased towards locality and certain canonical ANN topologies

The unified approach combines the algorithmic advances behind adaptive HyperNEAT and ES-HyperNEAT, each abstracting an important facet of natural evolution that contributes to its ability to evolve complexity. While ES-HyperNEAT and adaptive HyperNEAT have shown promise, it is the combination of the two methods that allows the evolution of complex regular plastic ANNs, which is a major goal for neuroevolution. Adaptive ES-HyperNEAT is able to fully determine the geometry, density, and plasticity of an evolving ANN. The results in a continuous T-Maze (Section 6.2) demonstrate that these capabilities combine effectively in a task that requires adaptation.

8.3 Future Directions

NE has produced successful results in a variety of domains but higher cognitive abilities remain to be evolved. One such ability that is important for both humans and animals is to localize and to navigate in an unknown environment. Ethological studies (O’Keefe and

Dostrovsky, 1971) suggest that rats and primates build an internal representation, i.e. a *cognitive map*, of the environment that allows goal-directed movement planning.

The cognitive map, hypothesized by Tolman (1948), allows rats not only to learn a specific path leading to a food item but also an internal representations of their environment that guides the local navigation. This hypothesis is supported by the discovery of *place cells* (O’Keefe and Dostrovsky, 1971) and *head direction cells* (Taube et al., 1990). Place cells are neurons found primarily in the hippocampus and their activation is correlated to the rat’s position in the environment. Once the rat enters a new environment, place cells become established within minutes and stay active even in the dark (Quirk et al., 1990). They tend to be stable in the same environment but when entering a new environment the place fields of neurons can become completely different, a process known as *remapping*. On the other hand, the activity of head direction cells depends on the orientation of the rat’s head while navigating.

Adaptive ES-HyperNEAT can potentially later be applied to such a neural-inspired navigation domain that requires map-based reasoning. An interesting question in such a domain is whether an architecture reminiscent of place cells, might evolve. If they did, or if an alternative structure emerged, it would provide insight into possible neural organizations for intelligent thought and significantly impact the wider research community. Neuro-ethological analyses of the evolved ANNs could potentially also deepen our understanding of how similar cognitive processes evolved in nature.

Future work for adaptive ES-HyperNEAT will also explore other challenging domains. While the work in this dissertation focused on relatively simple problems, it is an important future research question how close evolved indirectly encoded ANNs can approach the complexity of biological brains. For example, a challenging task would be to evolve a new generation of more brain-like adaptive controllers for legged robots. While Chapter 7 presented first steps towards that direction by establishing the benefits of learning a relationship between morphology and control, a plastic controller could potentially be deployed into an even wider range of body variants and seamlessly adapt to their bodies of origin, just as people can walk as they grow up through a wide array of body sizes and proportions. Another intriguing possibility is the evolution of a robot that starts crawling and then incrementally learns to walk in an upright posture, which would be a step towards creating a new generation of intelligent systems. Such research could help to facilitate the manufacturing of a variety of different robots because it would eliminate the need to design model-specific controllers.

CHAPTER 9

CONCLUSION

This dissertation introduced two extensions to a neuroevolutionary method called HyperNEAT, as well as their unification, adaptive ES-HyperNEAT, a new approach that allows the evolved ANNs to assume more brain-like properties. This chapter summarizes the main contributions of the dissertation.

9.1 Contributions

In summary, by showing that the ANNs evolved by adaptive ES-HyperNEAT assume more brain-like properties (e.g. topography, regularity, heterogeneous plasticity, and neuromodulation), this dissertation validates the research hypothesis that evolving more brain-like ANNs will allow neuroevolution to solve complex control problems and adaptive tasks more easily than current neuroevolution technology.

1. The advantages of evolving plastic neural networks with novelty search (Chapter 3) were established by analyzing the inherent deceptiveness in a variety of different dynamic, reward-based learning tasks. Because novelty search has the potential to foster the emergence of adaptive behavior in reward-based learning tasks, it opens a new direction for research in evolving adaptive ANNs.

2. Adaptive HyperNEAT extends HyperNEAT to allow not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of arbitrary learning rules (Chapter 4). The idea that learning rules can be distributed in a geometric pattern is new to neuroevolution but reflects the intuition that synaptic plasticity in biological brains is not encoded in DNA separately for every synapse in the brain. Thus the main idea behind adaptive ES-HyperNEAT is a step towards more biologically plausible adaptive systems.
3. HyperNEAT was extended to automatically determine the placement and density of the hidden nodes based on implicit information in the infinite-resolution pattern of weights (Chapter 5). The ES-HyperNEAT approach not only can evolve the location of every neuron in the network, but also can represent regions of varying density, which means resolution can increase holistically over evolution. ES-HyperNEAT was demonstrated through multi-task, maze navigation, and modular retina domains, revealing that the neural networks generated by this new approach assume more natural properties such as neural topography and geometric regularity. Also importantly, ES-HyperNEAT's compact indirect encoding can be seeded to begin with a bias towards a desired class of ANN topologies, which facilitates the evolutionary search. As a result, ES-HyperNEAT significantly outperforms the original HyperNEAT and expands the scope of neural structures that evolution can discover.

4. The combined approach, adaptive ES-HyperNEAT, unifies for the first time the ability to indirectly encode connectivity through geometry, simultaneously encode the density and placement of nodes in space, and generate patterns of heterogeneous plasticity (Chapter 6). In this way, adaptive ES-HyperNEAT takes another step towards more biologically-plausible ANNs. Importantly, the combined abilities of adaptive HyperNEAT and ES-HyperNEAT work well together, as demonstrated by experiments in the continuous T-Maze learning task.
5. Finally, a step towards the general goal of adaptive controllers for locomotion was taken by showing that HyperNEAT allows learning the relationship between morphology and control of different quadruped robots (Chapter 7). Once such a relationship is learned the controllers are able to work on a diversity of never-seen intermediate morphologies without any further training.

9.2 Conclusion

This dissertation presented two new extension to HyperNEAT called adaptive HyperNEAT and ES-HyperNEAT that overcome shortcomings of the original HyperNEAT approach. Adaptive HyperNEAT allows not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of arbitrary learning rules. The second HyperNEAT extension, called ES-HyperNEAT, is introduced to automatically

determines the placement and density of the hidden nodes and was shown to significantly outperform the original HyperNEAT in multiple domains. The combined approach, adaptive ES-HyperNEAT can for the first time compactly encode and discover neural geometry and plasticity that exhibit more natural features than traditional ANNs. The conclusion is that it is becoming possible to produce more natural neural structures through artificial evolution, which increasingly opens up opportunities for discovery relevant to the broader field of AI.

APPENDIX: PARAMETERS AND PSEUDOCODE

This appendix first describes the parameters and their chosen values across all the experiments in this dissertation and then gives pseudocode for the ES-HyperNEAT algorithm.

1. **Population Size:** The number of networks evaluated every generation.
2. c_1 : A parameter from the original NEAT that defines the weight of excess genes when computing compatibility between networks.
3. c_2 : A parameter from the original NEAT that defines the weight of disjoint genes when computing compatibility between networks.
4. c_3 : A parameter from the original NEAT that defines the weight of connection strength differences when computing compatibility between networks.
5. c_t : A parameter from the original NEAT that defines the compatibility threshold that determines whether networks are in the same species. In SharpNEAT and Hyper-SharpNEAT, this value is variable and changes to accommodate the desired number of species.
6. **Add Link Probability:** The probability of adding a new connection to a network (i.e. to the CPPN in case of HyperNEAT).
7. **Add Node Probability:** The probability of adding a new node to a network (i.e. to the CPPN in case of HyperNEAT).

8. **Target # of Species:** The desired number of species in the population. If there are more or less species, c_t will be adjusted down or up respectively in an effort to maintain the target number of species.
9. **Elitism:** The top percentage of each species that will be copied into the next generation unchanged.
10. **Expression Threshold:** The value for which a CPPN's output must exceed for a connection to be expressed in the original HyperNEAT.
11. **Function Set:** The set of functions that the CPPN can choose from when adding a node.
12. **Initial Resolution:** A parameter that defines the initial resolution for the division phase of ES-HyperNEAT.
13. **Maximum Resolution:** A parameter that defines the maximum resolution for the division phase of ES-HyperNEAT.
14. **Band Pruning Threshold:** The value that the band level of a connection must exceed to be expressed in ES-HyperNEAT.
15. **Variance Threshold:** The variance value that determines how far the depth-first search in the pruning and extraction phase in ES-HyperNEAT should traverse the quadtree.

16. **Iteration Level:** The parameter that determines how many times the quadtree extraction algorithm in ES-HyperNEAT should be iteratively applied to the discovered hidden neurons.
17. **Division Threshold:** The variance value that a quadtree node in ES-HyperNEAT must exceed to be further divided.

The NEAT coefficients c_1, c_2, c_3 were all set to 1.0 for all experiments. The novelty search experiments with NEAT (Chapter 3) were run with an extension of the steady-state real-time NEAT (rtNEAT) package (Stanley, 2006-2008). The population size was 500, with a 0.001 probability of adding a node and 0.01 probability of adding a link. The evaluation count for the single T-Maze domain was 125,000, and 950,000 for the double T-Maze and bee domain.

The adaptive HyperNEAT (Chapter 4), ES-HyperNEAT (Chapter 5) and adaptive ES-HyperNEAT (Chapter 6) experiments were run with the HyperSharpNEAT Simulator and Experimental Platform v1.0, which builds on a modified version of the public domain SharpNEAT package (Green, 2003–2006). The simulator and the ES-HyperNEAT source code can be found at <http://eplex.cs.ucf.edu/software.html>. Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters (Stanley and Miikkulainen, 2002). These parameters were found to be robust to moderate variation through preliminary experimentation. Elitism was set to 10%.

The quadruped experiments (Chapter 7) were run with a HyperNEAT extension of the steady-state rtNEAT package (Stanley, 2006-2008) in a realistic physics simulation using the freely available Open Dynamics Engine (see <http://www.ode.org>).

The parameters for all HyperNEAT experiments in this dissertation are shown in table A.1. The generation count and ES-HyperNEAT’s maximum resolution can vary from experiment to experiment depending on the complexity of the task. The HyperNEAT implementations based on the rtNEAT and SharpNEAT packages differ slightly in their implementations and their experiments were performed separately; accordingly they use slightly different parameters.

Table A.1: Parameter Settings in HyperNEAT Experiments

Parameter	Discrete T-Maze	Dual Task & Maze	Retina Task	Continues T-Maze	Quadruped Experiment
Population Size	500	300	300	300	200
Generation Count	500	500	2000	1000	600
Link Add. Prob.	0.03	0.03	0.03	0.03	0.06
Node Add. Prob.	0.02	0.02	0.02	0.02	0.005
Asexual Prob.	0.5	0.5	0.5	0.5	0.25
Sexual Prob.	0.5	0.5	0.5	0.5	0.75
Expr. Threshold	0.4	0.7	0.7	0.7	0.0
Initial Resolution	-	8×8	8×8	8×8	-
Max Resolution	-	8×8	32×32	16×16	-
Variance Threshold	-	0.03	0.03	0.03	-
Division Threshold	-	0.03	0.03	0.03	-
Band Threshold	-	0.3	0.3	0.3	-
Iteration Level	-	1	1	1	-
Function Set	Gaussian, Sine, Sigmoid, Abs. Value	Gaussian, Sine, Sigmoid, Abs. Value	Gaussian, Sine, Sigmoid, Abs. Value	Gaussian, Sine, Sigmoid, Abs. Value, Linear Step, Ramp	Gaussian, Sine, Sigmoid, Abs. Value, Cosine

Algorithm 1: DivisionAndInitialization(a, b , outgoing)

input : Coordinates of source (*outgoing = true*) or target node (*outgoing = false*) at (a, b).

output: Quadtree, in which each quadnode at (x, y) stores CPPN activation level for its position. The initialized quadtree is used in the PruningAndExtraction phase to generate the actual ANN connections.

```
1 begin
2    $root \leftarrow \text{QuadPoint}(0, 0, 1, 1)$  ; // x, y, width, level
3    $q \leftarrow \text{Queue}()$  ;
4    $q.\text{enqueue}(root)$  ;
5   while  $q$  is not empty do
6      $p \leftarrow q.\text{dequeue}()$ ;
7     // Divide into sub-regions and assign children to parent
8      $p.cs[0] \leftarrow \text{QuadPoint}(p.x - p.width/2, p.y - p.width/2, p.width/2, p.lev + 1)$  ;
9      $p.cs[1] \leftarrow \text{QuadPoint}(p.x - p.width/2, p.y + p.width/2, p.width/2, p.lev + 1)$  ;
10     $p.cs[2] \leftarrow \text{QuadPoint}(p.x + p.width/2, p.y + p.width/2, p.width/2, p.lev + 1)$  ;
11     $p.cs[3] \leftarrow \text{QuadPoint}(p.x + p.width/2, p.y - p.width/2, p.width/2, p.lev + 1)$  ;
12    foreach  $c \in p.cs$  do
13      if outgoing then // Querying connection from input or hidden node
14         $c.w \leftarrow \text{CPPN}(a, b, c.x, c.y)$ ; // Outgoing connectivity pattern
15      else // Querying connection to output node
16         $c.w \leftarrow \text{CPPN}(c.x, c.y, a, b)$ ; // Incoming connectivity pattern
17      end
18    end
19    // Divide until initial resolution or if variance is still high
20    if ( $p.level < initialDepth$ ) | ( $p.level < maxDepth$  &  $\text{variance}(p) > divThr$ ) then
21      foreach  $child \in p.cs$  do
22         $q.\text{enqueue}(child)$ ;
23      end
24    end
25  end
26  return  $root$ ;
27 end
```

Algorithm 2: PruningAndExtraction($a, b, \text{connections}, p, \text{outgoing}$)

input : Coordinates of source ($\text{outgoing} = \text{true}$) or target node ($\text{outgoing} = \text{false}$) at (a, b) and initialized quadtree p .
output: Adds the connections that are in bands of the two-dimensional cross-section of the hypercube containing the source or target node to the *connections* list.

```
1 begin
2   // Traverse quadtree depth-first
3   foreach  $c \in p.cs$  do
4     if  $\text{variance}(c) \geq \text{varianceThreshold}$  then
5       PruningAndExtraction( $a, b, \text{connections}, c, \text{outgoing}$ );
6     else
7       // Determine if point is in a band by checking neighbor CPPN
          values
8       if  $\text{outgoing}$  then
9          $d_{\text{left}} \leftarrow |c.\text{value} - \text{CPPN}(a, b, c.x-p.\text{width}, c.y)|$ ;
10         $d_{\text{right}} \leftarrow |c.\text{value} - \text{CPPN}(a, b, c.x+p.\text{width}, c.y)|$ ;
11         $d_{\text{top}} \leftarrow |c.\text{value} - \text{CPPN}(a, b, c.x, c.y-p.\text{width})|$ ;
12         $d_{\text{bottom}} \leftarrow |c.\text{value} - \text{CPPN}(a, b, c.x, c.y+p.\text{width})|$ ;
13      else // Querying connection to output node
14         $d_{\text{left}} \leftarrow |c.\text{value} - \text{CPPN}(c.x-p.\text{width}, c.y, a, b)|$ ;
15         $d_{\text{right}} \leftarrow |c.\text{value} - \text{CPPN}(c.x+p.\text{width}, c.y, a, b)|$ ;
16         $d_{\text{top}} \leftarrow |c.\text{value} - \text{CPPN}(c.x, c.y-p.\text{width}, a, b)|$ ;
17         $d_{\text{bottom}} \leftarrow |c.\text{value} - \text{CPPN}(c.x, c.y+p.\text{width}, a, b)|$ ;
18      end
19      if  $\max(\min(d_{\text{top}}, d_{\text{bottom}}), \min(d_{\text{left}}, d_{\text{right}})) > \text{bandThreshold}$  then
20        // Create new connection specified by  $(x1, y1, x2, y2, \text{weight})$ 
21        if  $\text{outgoing}$  then
22           $\text{con} \leftarrow \text{Connection}(a, b, c.x, c.y, c.w)$ ;
23        else // Querying connection to output node
24           $\text{con} \leftarrow \text{Connection}(c.x, c.y, a, b, c.w)$ ;
25        if  $\text{con} \notin \text{connections}$  then  $\text{connections} \leftarrow \text{connections} \cup \text{con}$ ;
26      end
27    end
28  end
29 end
```

Algorithm 3: ES-HyperNEAT

```
1 /* Parameters:  initialDepth, maxDepth, varianceThreshold,
   bandThreshold, iterationLevel, divisionThreshold */
   input : CPPN, InputPositions, OutputPositions
   output: Connections, HiddenNodes
2 begin
3   foreach input  $\in$  InputPositions do    // Input to hidden node connections
4     // Analyze outgoing connectivity pattern from this input
5     root  $\leftarrow$  DivisionAndInitialization (input.x, input.y, true);
6     // Traverse quadtree and add connections to list
7     PruningAndExtraction (input.x, input.y, connections1, root, true);
8     foreach c  $\in$  connections1 do
9       node  $\leftarrow$  Node(c.x2, c.y2);
10      if node  $\notin$  HiddenNodes then HiddenNodes  $\leftarrow$  HiddenNodes  $\cup$  node;
11    end
12  end
13  // Hidden to hidden node connections
14  UnexploredHiddenNodes  $\leftarrow$  HiddenNodes;
15  for i = 1 to iterationLevel do
16    foreach hidden  $\in$  UnexploredHiddenNodes do
17      root  $\leftarrow$  DivisionAndInitialization (hidden.x, hidden.y, true);
18      PruningAndExtraction (hidden.x, hidden.y, connections2, root, true);
19      foreach c  $\in$  connections2 do
20        node  $\leftarrow$  Node(c.x2, c.y2);
21        if node  $\notin$  HiddenNodes then HiddenNodes  $\leftarrow$  HiddenNodes  $\cup$  node;
22      end
23    end
24    // Remove the just explored nodes
25    UnexploredHiddenNodes  $\leftarrow$  HiddenNodes - UnexploredHiddenNodes;
26  end
27  foreach output  $\in$  OutputPositions do    // Hidden to output connections
28    // Analyze incoming connectivity pattern to this output
29    root  $\leftarrow$  DivisionAndInitialization (output.x, output.y, false);
30    PruningAndExtraction (output.x, output.y, connections3, root, false);
31    /* Nodes not created here because all the hidden nodes that are
       connected to an input/hidden node are already expressed. */
32  end
33  connections  $\leftarrow$  connections1  $\cup$  connections2  $\cup$  connections3;
34  Remove all neurons and their connections that do not have a path to an input and
  an output neuron;
35 end
```

LIST OF REFERENCES

- T. Aaltonen et al. Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. *Physical Review Letters*, 102(15):2001, 2009.
- P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65, 1994.
- M. Barlow, J. Galloway, and H. A. Abbass. Mining evolution through visualization. In E. Bilotta et al., editors, *ALife VIII Workshops*, pages 103–112, 2002.
- J. Baxter. The evolution of learning algorithms for artificial neural networks. In D. Green and T. Bossomaier, editors, *Complex Systems*, pages 313–326. IOS Press, 1992.
- M. Bear, B. Connors, and M. Paradiso. *Neuroscience: Exploring the brain*. Lippincott Williams & Wilkins, 2007.
- P. J. Bentley and S. Kumar. The ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 35–43, San Francisco, 1999. Kaufmann.
- J. Blynel and D. Floreano. Levels of Dynamics and Adaptive Behavior in Evolutionary Neural Controllers. In *From Animals to Animats 7*, 2002. In B. Hallam, D. Floreano, J. Hallam, G. Hayes, and J.-A. Meyer (eds).
- J. Blynel and D. Floreano. Exploring the T-Maze: Evolving learning-like robot behaviors using CTRNNs. In *2nd European Workshop on Evolutionary Robotics (EvoRob 2003)*, Lecture Notes in Computer Science, 2003.

- J. C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.
- R. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2):253–262, 1989.
- T. Carew, E. Walters, and E. Kandel. Classical conditioning in a simple withdrawal reflex in *Aplysia californica*. *The Journal of Neuroscience*, 1(12):1426–1437, 1981.
- D. J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 81–90. Morgan Kaufmann, 1990.
- J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Session on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.
- J. Clune, B. E. Beckmann, P. K. McKinley, and C. Ofria. Investigating whether HyperNEAT produces modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, New York, NY, 2010. ACM Press.
- J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *Evolutionary Computation, IEEE Transactions on*, 15(3):346–367, 2011.
- G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

- D. D'Ambrosio and K. O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.
- D. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '10*, pages 731–738, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- J. Drchal, J. Koutník, and M. Šnorek. HyperNEAT controlled robots learn to drive on roads in simulated environment. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*. IEEE Press, 2009.
- P. Dürri, C. Mattiussi, A. Soltoggio, and D. Floreano. Evolvability of neuromodulated learning for robots. In *The 2008 ECSIS Symposium on Learning and Adaptive Behavior in Robotic Systems*, pages 41–46, Los Alamitos, CA, 2008. IEEE Computer Society.
- P. Dürri, C. Mattiussi, and D. Floreano. Genetic representation and evolvability of modular neural controllers. *IEEE Computational Intelligence Magazine*, 2010.
- R. Dybowski, T. D. Collins, W. Hall, and P. R. Weller. Visualization of binary string convergence by sammon mapping. In *Proceedings of The Fifth Annual Conference on Evolutionary Programming*. MIT Press, 1996.
- H. Eichenbaum. A cortical–hippocampal system for declarative memory. *Nature Reviews Neuroscience*, 1(1):41–50, 2000.

- R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
- D. Floreano and J. Urzelai. Evolutionary robots with online self-organization and behavioral fitness. *Neural Networks*, 13:431–443, 2000.
- D. Floreano, P. Dürri, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
- J. Gauci and K. O. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press.
- J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Comput.*, 22:1860–1898, 2010.
- D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc. ISBN 0-8058-0158-8.
- F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1356–1361. Morgan Kaufmann, 1999.
- C. Green. SharpNEAT homepage. <http://sharpneat.sourceforge.net/>, 2003–2006.
- S. Grillner and P. Wallen. Central pattern generators for locomotion, with special reference to vertebrates. *Annual review of neuroscience*, 8(1):233–261, 1985.

- F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press, 1996.
- T. F. Hansen. Is modularity necessary for evolvability?: Remarks on the relationship between pleiotropy and evolvability. *Biosystems*, 69(2-3):83 – 94, 2003.
- I. Harvey. *The Artificial Evolution of Adaptive Behavior*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993.
- G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1, 1987.
- G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with AIBO. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 3040–3045, 2000.
- G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3), 2002.
- E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, Sept. 2004.
- L. P. Kaelbling, M. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence*, 4:237–285, May 1996.
- E. R. Kandel, J. H. Schwartz, and T. M. Jessell, editors. *Principles of Neural Science*. Elsevier, Amsterdam, third edition, 1991.

- N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773, 2005.
- Y.-H. Kim and B.-R. Moon. New Usage of Sammon’s Mapping for Genetic Visualization. In *Proceedings Genetic and Evolutionary Computation (GECCO 2003)*, pages 1136–1147. Springer-Verlag, 2003.
- H. Kimura, T. Yamashita, and S. Kobayashi. Reinforcement learning of walking behavior for a four-legged robot. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, pages 411–416, 2001.
- M. Kirschner and J. Gerhart. Evolvability. *Proceedings of the National Academy of Sciences of the United States of America*, 95(15):8420, 1998.
- J. Kittler and P. C. Young. A new approach to feature selection based on the Karhunen-Loeve expansion. *Pattern Recognition*, 5:335–352, 1973.
- J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In S. Bullock, J. Noble, R. Watson, and M. Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, Cambridge, MA, 2008. MIT Press.
- J. Lehman and K. O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO ’10, pages 103–110, New York, NY, USA, 2010a. ACM.

- J. Lehman and K. O. Stanley. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation, GECCO '10*, pages 837–844, New York, NY, USA, 2010b. ACM.
- J. Lehman and K. O. Stanley. Improving evolvability through novelty search and self-adaptation. pages 2693–2700, Piscataway, NJ, 2011a. IEEE Press.
- J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011b.
- M. A. Lewis and G. A. Bekey. Gait adaptation in a quadruped robot. *Autonomous Robots*, 12:301–312, 2002.
- H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(4):125, 2007. ISSN 1512-0856.
- W. W. Lytton and T. J. Sejnowski. Simulations of cortical pyramidal neurons synchronized by inhibitory interneurons. *Journal of Neurophysiology*, 66:1059–1079, 1991.
- A. P. Martin. Increasing genomic complexity by gene duplication and the origin of vertebrates. *The American Naturalist*, 154(2):111–128, 1999.
- G. Mayley. Guiding or hiding: Explorations into the effects of learning on the rate of evolution. In *Fourth European Conference on Artificial Life*, pages 135–144. MIT Press, 1997.
- G. McHale and P. Husbands. Gasnets and other evolvable neural networks applied to bipedal locomotion. *From Animals to Animats 8*, 2004.

- P. McQuesten and R. Miikkulainen. Culling and teaching in neuro-evolution. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, 1997. Kaufmann.
- J. Mouret. Novelty-based multiobjectivization. In *Proceedings of IROS Workshop on Exploring New Horizons in the Evolutionary Design of Robots*, 2009.
- Y. Niv, D. Joel, I. Meilijson, and E. Ruppin. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24, 2002. ISSN 1059-7123.
- S. Nolfi and D. Floreano. Learning and evolution. *Autonomous Robots*, 7(1):89–113, July 1999.
- S. Nolfi and D. Parisi. Auto-teaching: Networks that develop their own teaching input. In *Proceedings of the Second European Conference on Artificial Life*, pages 845–862., 1993.
- S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5:75–98, 1996.
- S. Nolfi, D. Parisi, and J. L. Elman. Learning and evolution in neural networks. *Adaptive Behavior*, 3:5–28, 1994.
- J. O’Keefe and J. Dostrovsky. The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 34(1):171, 1971.
- K. Pearson et al. The control of walking. *Scientific American*, 235(6):72, 1976.
- M. Pigliucci. Is evolvability evolvable? *Nature Reviews Genetics*, 9(1):75–82, 2008.

- G. Quirk, R. Muller, and J. Kubie. The firing of hippocampal place cells in the dark depends on the rat's recent experience. *Journal of Neuroscience*, 10(6):2008, 1990.
- T. Reil and P. Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, 6(2):159–168, April 2002.
- J. Reisinger, K. O. Stanley, and R. Miikkulainen. Towards an empirical measure of evolvability. In *Genetic and Evolutionary Computation Conference (GECCO2005) Workshop Program*, pages 257–264, Washington, D.C., 2005. ACM Press.
- S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In S. Doncieux, B. Girard, A. Guillot, J. Hallam, J.-A. Meyer, and J.-B. Mouret, editors, *From Animals to Animats 11*, volume 6226 of *Lecture Notes in Computer Science*, pages 533–543. Springer Berlin / Heidelberg, 2010.
- S. Risi and K. O. Stanley. Enhancing ES-HyperNEAT to evolve more complex regular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 1539–1546, New York, NY, USA, 2011. ACM.
- S. Risi and K. O. Stanley. A Unified Approach to Evolving Plasticity and Neural Geometry. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2012)*, Piscataway, NJ, 2012. IEEE Press. To appear.
- S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, New York, NY, USA, 2009. ACM Press.

- S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 18:470–491, December 2010a. ISSN 1059-7123.
- S. Risi, J. Lehman, and K. O. Stanley. Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY, 2010b. ACM Press.
- A. Rosenfeld. Quadrees and pyramids for pattern recognition and image processing. In *Proceedings of the 5th International Conference on Pattern Recognition*, pages 802–809. IEEE Press, 1980.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, pages 318–362. MIT Press, 1986.
- J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.*, 18(5):401–409, 1969. ISSN 0018-9340.
- N. Saravanan and D. B. Fogel. Evolving neural control systems. *IEEE Expert: Intelligent Systems and Their Applications*, 10(3):23–27, 1995. ISSN 0885-9000.
- J. Secretan, N. Beato, D. B. D’Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *CHI ’08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1759–1768, New York, NY, USA, 2008. ACM.
- J. Secretan, N. Beato, D. B. D.Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of

- design space. *Evolutionary Computation*, 2011. To appear.
- A. Soltoggio. Neural Plasticity and Minimal Topologies for Reward-Based Learning. In *Proceedings of the 2008 8th International Conference on Hybrid Intelligent Systems*, pages 637–642. IEEE Computer Society, 2008.
- A. Soltoggio, P. Dürri, C. Mattiussi, and D. Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007.
- A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürri, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Artificial Life XI*, pages 569–576, Cambridge, MA, 2008. MIT Press.
- O. Sporns. Network analysis, complexity, and brain function. *Complexity*, 8(1):56–60, 2002.
- URL <http://www3.interscience.wiley.com/cgi-bin/jissue/102525814>.
- K. O. Stanley. rtNEAT C++ software homepage: www.cs.utexas.edu/users/nn/keyword?rtneat. 2006-2008.
- K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162, 2007.
- K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.
- K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

- K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.
- K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC-2003)*. Canberra, Australia: IEEE Press, 2003.
- K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, December 2005.
- K. O. Stanley, D. B. D’Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.
- P. Strobach. Quadtree-structured recursive plane decomposition coding of images. *Signal Processing*, 39:1380–1397, 1991.
- R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998. ISBN 0262193981.
- J. Taube, R. Muller, and J. Ranck Jr. Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *Journal of Neuroscience*, 10(2):420, 1990.
- M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, July 2006.

- E. C. Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189–208, July 1948. ISSN 0033-295X.
- M. Van De Panne and A. Lamouret. Guided Optimization for Balanced Locomotion. In *6th Eurographics Workshop on Animation and Simulation*, Maastricht, Pays-Bas, 1995.
- V. K. Vassilev, T. C. Fogarty, and J. F. Miller. Information characteristics and the structure of landscapes. *Evol. Comput.*, 8(1):31–60, 2000. ISSN 1063-6560.
- P. Verbancsics and K. O. Stanley. Constraining Connectivity to Encourage Modularity in HyperNEAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, New York, NY, 2011. ACM.
- G. P. Wagner and L. Altenberg. Perspective: Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):pp. 967–976, 1996. ISSN 00143820.
- J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner. *Molecular Biology of the Gene Fourth Edition*. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.
- D. Wettergreen and C. Thorpe. Gait generation for legged robots. In *IEEE International Conference on Intelligent Robots and Systems*, 1992.
- S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *J. Mach. Learn. Res.*, 7:877–917, 2006. ISSN 1533-7928.
- B. Yamauchi and R. D. Beer. Integrating reactive, sequential, and learning behavior using dynamical neural networks. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference*

on Simulation of Adaptive Behavior, pages 382–391. Cambridge, MA: MIT Press, 1994.

ISBN 0-262-53122-4.

X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

M. J. Zigmond, F. E. Bloom, S. C. Landis, J. L. Roberts, and L. R. Squire, editors. *Fundamental Neuroscience*. Academic Press, London, 1999.